Oracle® Database

Database Performance Tuning Guide

19c
E96347-06
September 2022

Oracle Database Database Performance Tuning Guide, 19c

E96347-06

Contributing Authors: Glenn Maxey

Primary Authors: Rajesh Bhatiya, Immanuel Chan, Lance Ashdown

Contributors: Hermann Baer, Deba Chatterjee, Maria Colgan, Mikael Fries, Prabhaker Gongloor, Kevin Jernigan, Sue K. Lee, William Lee, David McDermid, Uri Shaft, Oscar Suro, Trung Tran, Sriram Vrinda, Yujun Wang

Contents

4 Configuring a Database for Performance

Part II Diagnosing and Tuning Database Performance

5 Measuring Database Performance

6 Gathering Database Statistics

7 Automatic Performance Diagnostics

8 Comparing Database Performance Over Time

10 Instance Tuning Using Performance Views

Part III Tuning Database Memory

11 Database Memory Allocation

12 Tuning the System Global Area

13 Tuning the Database Buffer Cache

14 Tuning the Shared Pool and the Large Pool

15 Tuning the Result Cache

16 Tuning the Program Global Area

18 Managing Operating System Resources

Preface

This preface contains these topics:

Audience

This document is intended for database administrators (DBAs) who are responsible for the operation, maintenance, and performance of Oracle Database. This guide describes how to use Oracle Database performance tools to optimize database performance. This guide also describes performance best practices for creating an initial database and includes performance-related reference information.

> **See Also:**
>
> - *Oracle Database SQL Tuning Guide* for information about how to optimize and tune SQL performance
> - *Oracle Database 2 Day + Performance Tuning Guide* to learn how to use Oracle Enterprise Manager Cloud Control (Cloud Control) to tune database performance

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

Before reading this guide, you should be familiar with the following documents:

- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day + Performance Tuning Guide*

To learn how to tune data warehouse environments, see *Oracle Database Data Warehousing Guide*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Database Performance Tuning Guide

This preface contains:

Changes in Oracle Database Release 19c, Version 19.1

The following are changes in *Oracle Database Performance Tuning Guide* for Oracle Database release 19c, version 19.1.

New Features

The following features are new in this release:

- Memoptimized Rowstore – Fast Ingest

  The fast ingest feature of the Memoptimized Rowstore optimizes the processing of high-frequency, single-row data inserts from applications, such as Internet of Things (IoT) applications.

  See "Using Fast Ingest".

- Automatic Database Diagnostic Monitor (ADDM) support for pluggable databases (PDBs)

  You can now use ADDM to analyze AWR data in PDBs for identifying and resolving performance related issues.

  See "Using ADDM in a Multitenant Environment".

Desupported Features

The following feature is desupported in this release.

- Oracle Streams

  Starting in Oracle Database 19c, the Oracle Streams feature is desupported. Use Oracle GoldenGate to replace all replication features of Oracle Streams.

Changes in Oracle Database Release 18c, Version 18.1

The following are changes in *Oracle Database Performance Tuning Guide* for Oracle Database release 18c, version 18.1.

New Features

The following features are new in this release:

- Memoptimized Rowstore

  The Memoptimized Rowstore enables high-performance reads for tables specified with the `MEMOPTIMIZE FOR READ` clause. This feature is particularly useful for the applications that mainly query tables based on primary key values at a very high frequency, such as Internet of Things (IoT) applications.

  See "Enabling High Performance Data Streaming with the Memoptimized Rowstore".

Changes in Oracle Database 12c Release 2 (12.2)

The following are changes in *Oracle Database Performance Tuning Guide* for Oracle Database 12*c* Release 2 (12.2).

New Features

The following features are new in this release:

- Per-process PGA limits

  A runaway query consuming excessive amount of PGA memory can create a serious performance problem in Oracle Database. In a multitenant container database (CDB), this type of query can affect the performance of all the pluggable databases (PDBs). To prevent this issue, you can now specify an absolute limit for the amount of PGA memory that can be used by each session in a particular consumer group.

  See "Sizing the Program Global Area Using the Resource Manager".

- Prespawned server processes

  Oracle Database now prespawns pools of server processes when dedicated broker connection mode is enabled or threaded execution mode is enabled. This feature improves the client connection performance in these modes of database operation.

  See "Improved Client Connection Performance Due to Prespawned Processes".

- Automatic Workload Repository (AWR) support for multitenant environment

  You can now capture Automatic Workload Repository (AWR) data for the PDBs in a multitenant environment. This feature enables performance tuning of PDBs in a multitenant environment.

  See "Managing Automatic Workload Repository in a Multitenant Environment".

- Automatic Workload Repository (AWR) support for Oracle Active Data Guard standby databases

You can now capture Automatic Workload Repository (AWR) data for Oracle Active Data Guard standby databases. This feature enables performance tuning of Oracle Active Data Guard standby databases.

See "Managing Automatic Workload Repository in Active Data Guard Standby Databases".

- Direct NFS Client performance improvement features - Parallel NFS and Direct NFS Dispatcher

  Parallel NFS is an optional feature of Oracle Direct NFS Client that is introduced in NFS version 4.1. This feature enables highly scalable distributed NAS storage for better I/O performance.

  The Direct NFS dispatcher feature consolidates the number of TCP connections that are created from a database instance to an NFS server. In large database deployments, using Direct NFS dispatcher improves scalability and network performance.

  See "Improving I/O Performance Using Direct NFS Client".

- Service-oriented buffer cache access optimization

  Cluster-managed services allocate workloads across various Oracle Real Application Clusters (Oracle RAC) database instances. These services also access data from the Oracle RAC database instances and provide it to the applications requesting the data. The Service-oriented Buffer Cache Access Optimization feature allows Oracle RAC to cache service-specific data that is frequently accessed by each of these services, thus improving the data access time for the services. This data-dependent caching also leads to more consistent response times when accessing data across Oracle RAC database instances. This feature is permanently enabled in Oracle RAC.

Changes in Oracle Database 12c Release 1 (12.1.0.2)

The following are changes in *Oracle Database Performance Tuning Guide* for Oracle Database 12*c* Release 1 (12.1.0.2).

New Features

The following features are new in this release:

- In-Memory Column Store

  The In-Memory Column Store (IM column store) is an optional area of the SGA that stores copies of tables, partitions, and other database objects in a columnar format that is optimized for rapid scans. IM column store accelerates database performance of analytics, data warehousing, and online transaction processing (OLTP) applications.

  See "Using the In-Memory Column Store to Improve Query Performance".

- Manageability support for In-Memory Column Store

  SQL Monitor report, ASH report, and AWR report now show statistics for various in-memory operations.

- Force full database caching mode

Force full database caching mode enables you to cache the entire database in memory, which may provide substantial performance improvements when performing full table scans or accessing LOBs.

See "Configuring the Database Caching Mode".

Changes in Oracle Database 12*c* Release 1 (12.1.0.1)

The following are changes in *Oracle Database Performance Tuning Guide* for Oracle Database 12*c* Release 1 (12.1.0.1).

New Features

The following features are new in this release:

- Real-Time ADDM

  Real-Time ADDM helps you to analyze and resolve problems in unresponsive or hung databases without having to restart the database.

  See "Real-Time ADDM Analysis".

- Limiting the size of the Program Global Area (PGA)

  The `PGA_AGGREGATE_LIMIT` initialization parameter enables you to specify a hard limit on PGA memory usage. Oracle Database ensures that the PGA size does not exceed this limit by terminating sessions or processes that are consuming the most PGA memory.

  See "Limiting the Size of the Program Global Area".

Other Changes

The following are additional changes in the release:

- New books

  Oracle Database Performance Tuning Guide is undergoing a major rewrite for this release. As a result, the book structure has undergone significant changes. This book now contains only tuning topics that pertain to the database itself. All SQL-related tuning topics are moved to *Oracle Database SQL Tuning Guide*.

Part I
Database Performance Fundamentals

This part contains the following chapters:

1

Performance Tuning Overview

This chapter provides an introduction to performance tuning and contains the following sections:

- Introduction to Performance Tuning
- Introduction to Performance Tuning Features and Tools

Introduction to Performance Tuning

This guide provides information about tuning Oracle Database for performance. Topics discussed in this guide include:

- Performance Planning
- Instance Tuning
- SQL Tuning

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* to learn how to use Oracle Enterprise Manager Cloud Control (Cloud Control) to tune database performance

Performance Planning

Refer to the topic Database Performance Fundamentals before proceeding with the other parts of this documentation. Based on years of designing and performance experience, Oracle has designed a performance methodology. This topic describes activities that can dramatically improve system performance, such as:

- Understanding Investment Options
- Understanding Scalability
- System Architecture
- Application Design Principles
- Workload Testing, Modeling, and Implementation
- Deploying New Applications

Instance Tuning

Diagnosing and Tuning Database Performance discusses the factors involved in the tuning and optimizing of an Oracle database instance.

When considering instance tuning, take care in the initial design of the database to avoid bottlenecks that could lead to performance problems. In addition, you must consider:

- Allocating memory to database structures
- Determining I/O requirements of different parts of the database
- Tuning the operating system for optimal performance of the database

After the database instance has been installed and configured, you must monitor the database as it is running to check for performance-related problems.

Performance Principles

Performance tuning requires a different, although related, method to the initial configuration of a system. Configuring a system involves allocating resources in an ordered manner so that the initial system configuration is functional.

Tuning is driven by identifying the most significant bottleneck and making the appropriate changes to reduce or eliminate the effect of that bottleneck. Usually, tuning is performed reactively, either while the system is in preproduction or after it is live.

Baselines

The most effective way to tune is to have an established performance baseline that you can use for comparison if a performance issue arises. Most database administrators (DBAs) know their system well and can easily identify peak usage periods. For example, the peak periods could be between 10.00am and 12.00pm and also between 1.30pm and 3.00pm. This could include a batch window of 12.00am midnight to 6am.

It is important to identify these peak periods at the site and install a monitoring tool that gathers performance data for those high-load times. Optimally, data gathering should be configured from when the application is in its initial trial phase during the QA cycle. Otherwise, this should be configured when the system is first in production.

Ideally, baseline data gathered should include the following:

- Application statistics (transaction volumes, response time)
- Database statistics
- Operating system statistics
- Disk I/O statistics
- Network statistics

In the Automatic Workload Repository, baselines are identified by a range of snapshots that are preserved for future comparisons. See "Automatic Workload Repository".

The Symptoms and the Problems

A common pitfall in performance tuning is to mistake the symptoms of a problem for the actual problem itself. It is important to recognize that many performance statistics indicate the symptoms, and that identifying the symptom is not sufficient data to implement a remedy. For example:

- Slow physical I/O

  Generally, this is caused by poorly-configured disks. However, it could also be caused by a significant amount of unnecessary physical I/O on those disks issued by poorly-tuned SQL.

- Latch contention

  Rarely is latch contention tunable by reconfiguring the instance. Rather, latch contention usually is resolved through application changes.

- Excessive CPU usage

  Excessive CPU usage usually means that there is little idle CPU on the system. This could be caused by an inadequately-sized system, by untuned SQL statements, or by inefficient application programs.

When to Tune

There are two distinct types of tuning:

- Proactive Monitoring
- Bottleneck Elimination

Proactive Monitoring

Proactive monitoring usually occurs on a regularly scheduled interval, where several performance statistics are examined to identify whether the system behavior and resource usage has changed. Proactive monitoring can also be considered as proactive tuning.

Usually, monitoring does not result in configuration changes to the system, unless the monitoring exposes a serious problem that is developing. In some situations, experienced performance engineers can identify potential problems through statistics alone, although accompanying performance degradation is usual.

Experimenting with or tweaking a system when there is no apparent performance degradation as a proactive action can be a dangerous activity, resulting in unnecessary performance drops. Tweaking a system should be considered reactive tuning, and the steps for reactive tuning should be followed.

Monitoring is usually part of a larger capacity planning exercise, where resource consumption is examined to see changes in the way the application is being used, and the way the application is using the database and host resources.

Bottleneck Elimination

Tuning usually implies fixing a performance problem. However, tuning should be part of the life cycle of an application—through the analysis, design, coding, production, and maintenance stages. Often, the tuning phase is left until the database is in production. At this time, tuning becomes a reactive process, where the most important bottleneck is identified and fixed.

Usually, the purpose for tuning is to reduce resource consumption or to reduce the elapsed time for an operation to complete. Either way, the goal is to improve the effective use of a particular resource. In general, performance problems are caused by the overuse of a particular resource. The overused resource is the bottleneck in the system. There are several distinct phases in identifying the bottleneck and the potential fixes. These are discussed in the sections that follow.

Remember that the different forms of contention are symptoms that can be fixed by making changes in the following places:

- Changes in the application, or the way the application is used
- Changes in Oracle
- Changes in the host hardware configuration

Often, the most effective way of resolving a bottleneck is to change the application.

SQL Tuning

Many application programmers consider SQL a messaging language, because queries are issued and data is returned. However, client tools often generate inefficient SQL statements. Therefore, a good understanding of the database SQL processing engine is necessary for writing optimal SQL. This is especially true for high transaction processing systems.

Typically, SQL statements issued by online transaction processing (OLTP) applications operate on relatively few rows at a time. If an index can point to the exact rows that are required, then Oracle Database can construct an accurate plan to access those rows efficiently through the shortest possible path. In decision support system (DSS) environments, selectivity is less important, because they often access most of a table's rows. In such situations, full table scans are common, and indexes are not even used. This book is primarily focussed on OLTP applications.

> **See Also:**
>
> - *Oracle Database SQL Tuning Guide* for detailed information on the process of tuning and optimizing SQL statements
> - *Oracle Database Data Warehousing Guide* for detailed information on decision support systems (DSS) and mixed environments

Query Optimizer and Execution Plans

When a SQL statement is executed on an Oracle database, the query optimizer determines the most efficient execution plan after considering many factors related to the objects referenced and the conditions specified in the query. This determination is an important step in the processing of any SQL statement and can greatly affect execution time.

During the evaluation process, the query optimizer reviews statistics gathered on the system to determine the best data access path and other considerations. You can override the execution plan of the query optimizer with hints inserted in SQL statement.

Introduction to Performance Tuning Features and Tools

Effective data collection and analysis is essential for identifying and correcting performance problems. Oracle Database provides several tools that allow a performance engineer to gather information regarding database performance. In

addition to gathering data, Oracle Database provides tools to monitor performance, diagnose problems, and tune applications.

The Oracle Database gathering and monitoring features are mainly automatic, managed by Oracle background processes. To enable automatic statistics collection and automatic performance features, the `STATISTICS_LEVEL` initialization parameter must be set to `TYPICAL` or `ALL`. You can administer and display the output of the gathering and tuning tools with Oracle Enterprise Manager Cloud Control (Cloud Control), or with APIs and views. For ease of use and to take advantage of its numerous automated monitoring and diagnostic tools, Cloud Control is recommended.

See Also:

- *Oracle Database 2 Day DBA* to learn how to use Cloud Control to manage Oracle Database
- *Oracle Database 2 Day + Performance Tuning Guide* to learn how to use Cloud Control to tune database performance
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information on the `DBMS_ADVISOR`, `DBMS_SQLTUNE`, `DBMS_AUTO_SQLTUNE`, and `DBMS_WORKLOAD_REPOSITORY` packages
- *Oracle Database Reference* for information about the `STATISTICS_LEVEL` initialization parameter

Automatic Performance Tuning Features

The Oracle Database automatic performance tuning features include:

- Automatic Workload Repository (AWR) collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. See "Automatic Workload Repository".
- Automatic Database Diagnostic Monitor (ADDM) analyzes the information collected by AWR for possible performance problems with the Oracle database. See "Overview of the Automatic Database Diagnostic Monitor".
- SQL Tuning Advisor allows a quick and efficient technique for optimizing SQL statements without modifying any statements. See *Oracle Database SQL Tuning Guide*.
- SQL Access Advisor provides advice on materialized views, indexes, and materialized view logs. See *Oracle Database SQL Tuning Guide*.
- End to End Application tracing identifies excessive workloads on the system by specific user, service, or application component. See *Oracle Database SQL Tuning Guide*.
- Server-generated alerts automatically provide notifications when impending problems are detected. See *Oracle Database Administrator's Guide* to learn how to monitor the operation of the database with server-generated alerts.
- Additional advisors that can be launched from Oracle Enterprise Manager Cloud Control (Cloud Control), such as memory advisors to optimize memory for an instance. The memory advisors are commonly used when automatic memory management is not set up for the database. Other advisors are used to optimize mean time to recovery (MTTR),

shrinking of segments, and undo tablespace settings. To learn about the advisors available with Cloud Control, see *Oracle Database 2 Day + Performance Tuning Guide*.

- The Database Performance page in Cloud Control displays host, instance service time, and throughput information for real time monitoring and diagnosis. The page can be set to refresh automatically in selected intervals or manually. To learn about the Database Performance page, see *Oracle Database 2 Day + Performance Tuning Guide*.

Additional Oracle Database Tools

This section describes additional Oracle Database tools that you can use for determining performance problems.

V$ Performance Views

The `V$` views are the performance information sources used by all Oracle Database performance tuning tools. The `V$` views are based on memory structures initialized at instance startup. The memory structures, and the views that represent them, are automatically maintained by Oracle Database for the life of the instance.

> **Note:**
>
> Oracle recommends using the Automatic Workload Repository to gather performance data. These tools have been designed to capture all of the data needed for performance analysis.

> **See Also:**
>
> - " Instance Tuning Using Performance Views" for more information about diagnosing database performance problems using the `V$` performance views
> - *Oracle Database Reference* for more information about dynamic performance views

2
Designing and Developing for Performance

Optimal system performance begins with design and continues throughout the life of your system. Carefully consider performance issues during the initial design phase so that you can tune your system more easily during production.

This chapter contains the following sections:

Oracle Methodology

System performance has become increasingly important as computer systems get larger and more complex as the Internet plays a bigger role in business applications. To accommodate this, Oracle has produced a performance methodology based on years of designing and performance experience. This methodology explains clear and simple activities that can dramatically improve system performance.

Performance strategies vary in their effectiveness, and systems with different purposes—such as operational systems and decision support systems—require different performance skills. This book examines the considerations that any database designer, administrator, or performance expert should focus their efforts on.

System performance is designed and built into a system. It does not just happen. Performance problems are usually the result of contention for, or exhaustion of, some system resource. When a system resource is exhausted, the system cannot scale to higher levels of performance. This new performance methodology is based on careful planning and design of the database, to prevent system resources from becoming exhausted and causing down-time. By eliminating resource conflicts, systems can be made scalable to the levels required by the business.

Understanding Investment Options

With the availability of relatively inexpensive, high-powered processors, memory, and disk drives, there is a temptation to buy more system resources to improve performance. In many situations, new CPUs, memory, or more disk drives can indeed provide an immediate performance improvement. However, any performance increases achieved by adding hardware should be considered a short-term relief to an immediate problem. If the demand and load rates on the application continue to grow, then the chance of the same problem occurring soon is likely.

In other situations, additional hardware does not improve the system's performance at all. Poorly designed systems perform poorly no matter how much extra hardware is allocated. Before purchasing additional hardware, ensure that serialization or single threading is not occurring within the application. Long-term, it is generally more valuable to increase the efficiency of your application in terms of the number of physical resources used for each business transaction.

Understanding Scalability

The word *scalability* is used in many contexts in development environments. The following section provides an explanation of scalability that is aimed at application designers and performance specialists.

This section covers the following topics:

What is Scalability?

Scalability is a system's ability to process more workload, with a proportional increase in system resource usage.

In a scalable system, if you double the workload, then the system uses twice as many system resources. This sounds obvious, but due to conflicts within the system, the resource usage might exceed twice the original workload.

Examples of poor scalability due to resource conflicts include the following:

- Applications requiring significant concurrency management as user populations increase
- Increased locking activities
- Increased data consistency workload
- Increased operating system workload
- Transactions requiring increases in data access as data volumes increase
- Poor SQL and index design resulting in a higher number of logical I/Os for the same number of rows returned
- Reduced availability, because database objects take longer to maintain

An application is said to be unscalable if it exhausts a system resource to the point where no more throughput is possible when its workload is increased. Such applications result in fixed throughputs and poor response times.

Examples of resource exhaustion include the following:

- Hardware exhaustion
- Table scans in high-volume transactions causing inevitable disk I/O shortages
- Excessive network requests, resulting in network and scheduling bottlenecks
- Memory allocation causing paging and swapping
- Excessive process and thread allocation causing operating system thrashing

This means that application designers must create a design that uses the same resources, regardless of user populations and data volumes, and does not put loads on the system resources beyond their limits.

System Scalability

Applications that are accessible through the Internet have more complex performance and availability requirements.

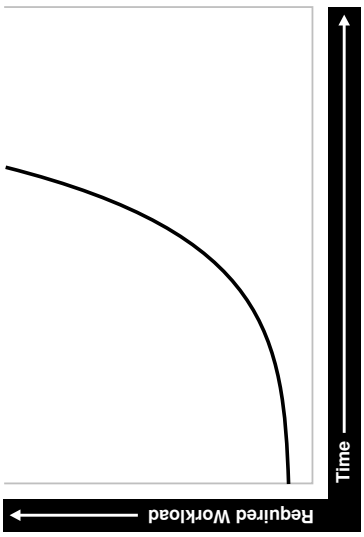
Some applications are designed and written only for Internet use, but even typical back-office applications—such as a general ledger application—might require some or all data to be available online.

Characteristics of Internet age applications include the following:

- Availability 24 hours a day, 365 days a year
- Unpredictable and imprecise number of concurrent users
- Difficulty in capacity planning
- Availability for any type of query
- Multitier architectures
- Stateless middleware
- Rapid development timescale
- Minimal time for testing

The following figure illustrates the classic workload growth curve, with demand growing at an increasing rate. Applications must scale with the increase of workload and also when additional hardware is added to support increasing demand. Design errors can cause the implementation to reach its maximum, regardless of additional hardware resources or re-design efforts.

Figure 2-1 Workload Growth Curve



Applications are challenged by very short development timeframes with limited time for testing and evaluation. However, bad design typically means that you must later rearchitect

and reimplement the system. If you deploy an application with known architectural and implementation limitations on the Internet, and if the workload exceeds the anticipated demand, then failure is a real possibility. From a business perspective, poor performance can mean a loss of customers. If Web users do not get a response in seven seconds, then the user's attention could be lost forever.

In many cases, the cost of re-designing a system with the associated downtime costs in migrating to new implementations exceeds the costs of properly building the original system. The moral of the story is simple: design and implement with scalability in mind from the start.

Factors Preventing Scalability

When building applications, designers and architects should aim for as close to perfect scalability as possible. This is sometimes called *linear* scalability, where system throughput is directly proportional to the number of CPUs.

In the real world, linear scalability is impossible for reasons beyond a designer's control. However, making the application design and implementation as scalable as possible should ensure that current and future performance objectives can be achieved through expansion of hardware components and the evolution of CPU technology.

Factors that may prevent linear scalability include:

- Poor application design, implementation, and configuration

  The application has the biggest impact on scalability. For example:

  - Poor schema design can cause expensive SQL that do not scale.
  - Poor transaction design can cause locking and serialization problems.
  - Poor connection management can cause poor response times and unreliable systems.

  However, the design is not the only problem. The physical implementation of the application can be the weak link. For example:

  - Systems can move to production environments with bad I/O strategies.
  - The production environment could might different execution plans from those generated in testing.
  - Memory-intensive applications that allocate a large amount of memory without much thought for freeing the memory at run time can cause excessive memory usage.
  - Inefficient memory usage and memory leaks put a high stress on the operating virtual memory subsystem. This impacts performance and availability.

- Incorrect sizing of hardware components

  Bad capacity planning of all hardware components is becoming less of a problem as relative hardware prices decrease. However, too much capacity can mask scalability problems as the workload is increased on a system.

- Limitations of software components

  All software components have scalability and resource usage limitations. This applies to application servers, database servers, and operating systems.

Application design should not place demands on the software beyond what it can handle.

- Limitations of hardware components

  Hardware is not perfectly scalable. Most multiprocessor computers can get close to linear scaling with a finite number of CPUs, but after a certain point each additional CPU can increase performance overall, but not proportionately. There might come a time when an additional CPU offers no increase in performance, or even degrades performance. This behavior is very closely linked to the workload and the operating system setup.

> **Note:**
>
> These factors are based on Oracle Server Performance group's experience of tuning unscalable systems.

System Architecture

There are two main parts to a system's architecture:

- Hardware and Software Components
- Configuring the Right System Architecture for Your Requirements

Hardware and Software Components

A system architecture mainly contains hardware and software components.

- Hardware Components
- Software Components

Hardware Components

Today's designers and architects are responsible for sizing and capacity planning of hardware at each tier in a multitier environment. It is the architect's responsibility to achieve a balanced design. This is analogous to a bridge designer who must consider all the various payload and structural requirements for the bridge. A bridge is only as strong as its weakest component. As a result, a bridge is designed in balance, such that all components reach their design limits simultaneously.

The following are the main hardware components of a system.

CPU

There can be one or more CPUs, and they can vary in processing power from simple CPUs found in hand-held devices to high-powered server CPUs. Sizing of other hardware components is usually a multiple of the CPUs on the system.

Memory

Database and application servers require considerable amounts of memory to cache data and avoid time-consuming disk access.

I/O Subsystem

The I/O subsystem can vary between the hard disk on a client PC and high performance disk arrays. Disk arrays can perform thousands of I/Os each second and provide availability through redundancy in terms of multiple I/O paths and hot pluggable mirrored disks.

Network

All computers in a system are connected to a network, from a modem line to a high speed internal LAN. The primary concerns with network specifications are bandwidth (volume) and latency (speed).

Software Components

The same way computers have common hardware components, applications have common functional components. By dividing software development into functional components, it is possible to better comprehend the application design and architecture. Some components of the system are performed by existing software bought to accelerate application implementation, or to avoid re-development of common components.

The difference between software components and hardware components is that while hardware components only perform one task, a piece of software can perform the roles of various software components. For example, a disk drive only stores and retrieves data, but a client program can manage the user interface and perform business logic.

Most applications involve the following software components:

User Interface

This component is the most visible to application users, and includes the following functions:

- Displaying the screen to the user
- Collecting user data and transferring it to business logic
- Validating data entry
- Navigating through levels or states of the application

Business Logic

This component implements core business rules that are central to the application function. Errors made in this component can be very costly to repair. This component is implemented by a mixture of declarative and procedural approaches. An example of a declarative activity is defining unique and foreign keys. An example of procedure-based logic is implementing a discounting strategy.

Common functions of this component include:

- Moving a data model to a relational table structure
- Defining constraints in the relational table structure
- Coding procedural logic to implement business rules

Resources for Managing User Requests

This component is implemented in all pieces of software. However, there are some requests and resources that can be influenced by the application design and some that cannot.

In a multiuser application, most resource allocation by user requests are handled by the database server or the operating system. However, in a large application where the number of users and their usage pattern is unknown or growing rapidly, the system architect must be proactive to ensure that no single software component becomes overloaded and unstable.

Common functions of this component include:

- Connection management with the database
- Executing SQL efficiently (cursors and SQL sharing)
- Managing client state information
- Balancing the load of user requests across hardware resources
- Setting operational targets for hardware and software components
- Persistent queuing for asynchronous execution of tasks

Data and Transactions

This component is largely the responsibility of the database server and the operating system.

Common functions of this component include:

- Providing concurrent access to data using locks and transactional semantics
- Providing optimized access to the data using indexes and memory cache
- Ensuring that data changes are logged in the event of a hardware failure
- Enforcing any rules defined for the data

Configuring the Right System Architecture for Your Requirements

Configuring the initial system architecture is a largely iterative process. System architects must satisfy the system requirements within budget and schedule constraints. If the system requires interactive users transacting business-making decisions based on the contents of a database, then user requirements drive the architecture. If there are few interactive users on the system, then the architecture is process-driven.

Examples of interactive user applications:

- Accounting and bookkeeping applications
- Order entry systems
- Email servers
- Web-based retail applications
- Trading systems

Examples of process-driven applications:

- Utility billing systems
- Fraud detection systems

- Direct mail

In many ways, process-driven applications are easier to design than multiuser applications because the user interface element is eliminated. However, because the objectives are process-oriented, system architects not accustomed to dealing with large data volumes and different success factors can become confused. Process-driven applications draw from the skills sets used in both user-based applications and data warehousing. Therefore, this book focuses on evolving system architectures for interactive users.

> **Note:**
>
> Generating a system architecture is not a deterministic process. It requires careful consideration of business requirements, technology choices, existing infrastructure and systems, and actual physical resources, such as budget and manpower.

The following questions should stimulate thought on system architecture, though they are not a definitive guide to system architecture. These questions demonstrate how business requirements can influence the architecture, ease of implementation, and overall performance and availability of a system. For example:

- How many users must the system support?

  Most applications fall into one of the following categories:

  - Very few users on a lightly-used or exclusive computer

    For this type of application, there is usually one user. The focus of the application design is to make the single user as productive as possible by providing good response time, yet make the application require minimal administration. Users of these applications rarely interfere with each other and have minimal resource conflicts.

  - A medium to large number of users in a corporation using shared applications

    For this type of application, the users are limited by the number of employees in the corporation actually transacting business through the system. Therefore, the number of users is predictable. However, delivering a reliable service is crucial to the business. The users must share a resource, so design efforts must address response time under heavy system load, escalation of resource for each session usage, and room for future growth.

  - An infinite user population distributed on the Internet

    For this type of application, extra engineering effort is required to ensure that no system component exceeds its design limits. This creates a bottleneck that halts or destabilizes the system. These applications require complex load balancing, stateless application servers, and efficient database connection management. In addition, use statistics and governors to ensure that the user receives feedback if the database cannot satisfy their requests because of system overload.

- What will be the user interaction method?

  The choices of user interface range from a simple Web browser to a custom client program.

- Where are the users located?

  The distance between users influences how the application is engineered to cope with network latencies. The location also affects which times of the day are busy, when it is impossible to perform batch or system maintenance functions.

- What is the network speed?

  Network speed affects the amount of data and the conversational nature of the user interface with the application and database servers. A highly conversational user interface can communicate with back-end servers on every key stroke or field level validation. A less conversational interface works on a screen-sent and a screen-received model. On a slow network, it is impossible to achieve high data entry speeds with a highly conversational user interface.

- How much data will the user access, and how much of that data is largely read only?

  The amount of data queried online influences all aspects of the design, from table and index design to the presentation layers. Design efforts must ensure that user response time is not a function of the size of the database. If the application is largely read only, then replication and data distribution to local caches in the application servers become a viable option. This also reduces workload on the core transactional server.

- What is the user response time requirement?

  Consideration of the user type is important. If the user is an executive who requires accurate information to make split second decisions, then user response time cannot be compromised. Other types of users, such as users performing data entry activities, might not need such a high level of performance.

- Do users expect 24 hour service?

  This is mandatory for today's Internet applications where trade is conducted 24 hours a day. However, corporate systems that run in a single time zone might be able to tolerate after-hours downtime. You can use this after-hours downtime to run batch processes or to perform system administration. In this case, it might be more economic not to run a fully-available system.

- Must all changes be made in real time?

  It is important to determine whether transactions must be executed within the user response time, or if they can be queued for asynchronous execution.

The following are secondary questions, which can also influence the design, but really have more impact on budget and ease of implementation. For example:

- How big will the database be?

  This influences the sizing of the database server. On servers with a very large database, it might be necessary to have a bigger computer than dictated by the workload. This is because the administration overhead with large databases is largely a function of the database size. As tables and indexes grow, it takes proportionately more CPUs to allow table reorganizations and index builds to complete in an acceptable time limit.

- What is the required throughput of business transactions?
- What are the availability requirements?
- Do skills exist to build and administer this application?
- What compromises are forced by budget constraints?

Application Design Principles

This section describes the following design decisions that are involved in building applications:

Simplicity In Application Design

Applications are no different than any other designed and engineered product. Well-designed structures, computers, and tools are usually reliable, easy to use and maintain, and simple in concept. In the most general terms, if the design looks correct, then it probably is. This principle should always be kept in mind when building applications.

Consider the following design issues:

- If the table design is so complicated that nobody can fully understand it, then the table is probably poorly designed.
- If SQL statements are so long and involved that it would be impossible for any optimizer to effectively optimize it in real time, then there is probably a bad statement, underlying transaction, or table design.
- If there are indexes on a table and the same columns are repeatedly indexed, then there is probably a poor index design.
- If queries are submitted without suitable qualification for rapid response for online users, then there is probably a poor user interface or transaction design.
- If the calls to the database are abstracted away from the application logic by many layers of software, then there is probably a bad software development method.

Data Modeling

Data modeling is important to successful relational application design. You must perform this modeling in a way that quickly represents the business practices. Heated debates may occur about the correct data model. The important thing is to apply greatest modeling efforts to those entities affected by the most frequent business transactions. In the modeling phase, there is a great temptation to spend too much time modeling the non-core data elements, which results in increased development lead times. Use of modeling tools can then rapidly generate schema definitions and can be useful when a fast prototype is required.

Table and Index Design

Table design is largely a compromise between flexibility and performance of core transactions. To keep the database flexible and able to accommodate unforeseen workloads, the table design should be very similar to the data model, and it should be normalized to at least 3rd normal form. However, certain core transactions required by users can require selective denormalization for performance purposes.

Examples of this technique include storing tables pre-joined, the addition of derived columns, and aggregate values. Oracle Database provides numerous options for storage of aggregates and pre-joined data by clustering and materialized view functions. These features allow a simpler table design to be adopted initially.

Again, focus and resources should be spent on the business critical tables, so that optimal performance can be achieved. For non-critical tables, shortcuts in design can be adopted to enable a more rapid application development. However, if prototyping and testing a non-core table becomes a performance problem, then remedial design effort should be applied immediately.

Index design is also a largely iterative process, based on the SQL generated by application designers. However, it is possible to make a sensible start by building indexes that enforce primary key constraints and indexes on known access patterns, such as a person's name. As the application evolves, and as you perform testing on realistic amounts of data, you may need to improve the performance of specific queries by building a better index. Consider the following list of indexing design ideas when building a new index:

- Appending Columns to an Index or Using Index-Organized Tables
- Using a Different Index Type
- Finding the Cost of an Index
- Serializing within Indexes
- Ordering Columns in an Index

Appending Columns to an Index or Using Index-Organized Tables

One of the easiest ways to speed up a query is to reduce the number of logical I/Os by eliminating a table access from the execution plan. This can be done by appending to the index all columns referenced by the query. These columns are the select list columns, and any required join or sort columns. This technique is particularly useful in speeding up online applications response times when time-consuming I/Os are reduced. This is best applied when testing the application with properly sized data for the first time.

The most aggressive form of this technique is to build an index-organized table (IOT). However, you must be careful that the increased leaf size of an IOT does not undermine the efforts to reduce I/O.

Using a Different Index Type

There are several index types available, and each index has benefits for certain situations. The following list gives performance ideas associated with each index type.

B-Tree Indexes

These indexes are the standard index type, and they are excellent for primary key and highly-selective indexes. Used as concatenated indexes, the database can use B-tree indexes to retrieve data sorted by the index columns.

Bitmap Indexes

These indexes are suitable for columns that have a relatively low number of distinct values, where the benefit of adding a B-tree index is likely to be limited. These indexes are suitable for data warehousing applications where there is low DML activity and ad hoc filtering patterns. Combining bitmap indexes on columns allows efficient `AND` and `OR` operations with minimal I/O. Further, through compression techniques they can generate a large number of rowids with minimal I/Os. Bitmap indexes are particularly efficient in queries with `COUNT()`, because the query can be satisfied within the index.

Function-based Indexes

These indexes allow access through a B-tree on a value derived from a function on the base data. Function-based indexes have some limitations with regards to the use of nulls, and they require that you have the query optimizer enabled.

Function-based indexes are particularly useful when querying on composite columns to produce a derived result or to overcome limitations in the way data is stored in the database. An example is querying for line items in an order exceeding a certain value derived from (sales price - discount) x quantity, where these were columns in the table. Another example is to apply the `UPPER` function to the data to allow case-insensitive searches.

Partitioned Indexes

Partitioning a global index allows partition pruning to take place within an index access, which results in reduced I/Os. By definition of good range or list partitioning, fast index scans of the correct index partitions can result in very fast query times.

Reverse Key Indexes

These indexes are designed to eliminate index hot spots on insert applications. These indexes are excellent for insert performance, but they are limited because the database cannot use them for index range scans.

Finding the Cost of an Index

Building and maintaining an index structure can be expensive, and it can consume resources such as disk space, CPU, and I/O capacity. Designers must ensure that the benefits of any index outweigh the negatives of index maintenance.

Use this simple estimation guide for the cost of index maintenance: each index maintained by an `INSERT`, `DELETE`, or `UPDATE` of the indexed keys requires about three times as much resource as the actual DML operation on the table. Thus, if you `INSERT` into a table with three indexes, then the insertion is approximately 10 times slower than an `INSERT` into a table with no indexes. For DML, and particularly for `INSERT`-heavy applications, the index design should be seriously reviewed, which might require a compromise between the query and `INSERT` performance.

> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn how to monitor index usage

Serializing within Indexes

Use of sequences or timestamps to generate key values that are indexed themselves can lead to database hotspot problems, which affect response time and throughput. This is usually the result of a monotonically growing key that results in a right-growing index. To avoid this problem, try to generate keys that insert over the full range of the index so as to make a workload more scalable. You can achieve this by using any of the following methods:

- using a reverse key index
- using a hash partitioned index
- using a cycling sequence to prefix sequence values
- using a scalable sequence

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about the scalable sequences

Ordering Columns in an Index

Designers should be flexible in defining any rules for index building. Depending on your circumstances, use one of the following two ways to order the keys in an index:

- Order columns with most selectivity first. This method is the most commonly used because it provides the fastest access with minimal I/O to the actual rowids required. This technique is used mainly for primary keys and for very selective range scans.
- Order columns to reduce I/O by clustering or sorting data. In large range scans, I/Os can usually be reduced by ordering the columns in the least selective order, or in a manner that sorts the data in the way it should be retrieved.

Using Views

Views can speed up and simplify application design. A simple view definition can mask data model complexity from the programmers whose priorities are to retrieve, display, collect, and store data.

However, while views provide clean programming interfaces, they can cause sub-optimal, resource-intensive queries. The worst type of view use is when a view references other views, and when they are joined in queries. In many cases, developers can satisfy the query directly from the table without using a view. Usually, because of their inherent properties, views make it difficult for the optimizer to generate the optimal execution plan.

SQL Execution Efficiency

In the design and architecture phase of any system development, care should be taken to ensure that the application developers understand SQL execution efficiency. To achieve this goal, the development environment must support the following characteristics:

- Good database connection management

  Connecting to the database is an expensive operation that is highly unscalable. Therefore, the number of concurrent connections to the database should be minimized as much as possible. A simple system, where a user connects at application initialization, is ideal. However, in a Web-based or multitiered application, where application servers are used to multiplex database connections to users, this can be difficult. With these types of applications, design efforts should ensure that database connections are pooled and are not reestablished for each user request.

- Good cursor usage and management

  Maintaining user connections is equally important to minimizing the parsing activity on the system. Parsing is the process of interpreting a SQL statement and creating an execution plan for it. This process has many phases, including syntax checking, security checking, execution plan generation, and loading shared structures into the shared pool. There are two types of parse operations:

  – Hard parsing

    A SQL statement is submitted for the first time, and no match is found in the shared pool. Hard parses are the most resource-intensive and unscalable, because they perform all the operations involved in a parse.

  – Soft parsing

    A SQL statement is submitted for the first time, and a match *is* found in the shared pool. The match can be the result of previous execution by another user. The SQL statement is shared, which is good for performance. However, soft parses are not ideal, because they still require syntax and security checking, which consume system resources.

  Because parsing should be minimized as much as possible, application developers should design their applications to parse SQL statements once and execute them many times. This is done through cursors. Experienced SQL programmers should be familiar with the concept of opening and re-executing cursors.

  Application developers must also ensure that SQL statements are shared within the shared pool. To achieve this goal, use bind variables to represent the parts of the query that change from execution to execution. If this is not done, then the SQL statement is likely to be parsed once and never re-used by other users. To ensure that SQL is shared, use bind variables and do not use string literals with SQL statements. For example:

  Statement with string literals:

```
SELECT * FROM employees
  WHERE last_name LIKE 'KING';
```

  Statement with bind variables:

```
SELECT * FROM employees
  WHERE last_name LIKE :1;
```

The following example shows the results of some tests on a simple OLTP application:

```
Test                          #Users Supported
No Parsing all statements           270
Soft Parsing all statements         150
Hard Parsing all statements          60
Re-Connecting for each Transaction   30
```

These tests were performed on a four-CPU computer. The differences increase as the number of CPUs on the system increase.

Implementing the Application

The choice of development environment and programming language is largely a function of the skills available in the development team and architectural decisions made when specifying the application. There are, however, some simple performance management rules that can lead to scalable, high-performance applications.

1. Choose a development environment suitable for software components, and do not let it limit your design for performance decisions. If it does, then you probably chose the wrong language or environment.
   - User interface

     The programming model can vary between HTML generation and calling the windowing system directly. The development method should focus on response time of the user interface code. If HTML or Java is being sent over a network, then try to minimize network volume and interactions.
   - Business logic

     Interpreted languages, such as Java and PL/SQL, are ideal to encode business logic. They are fully portable, which makes upgrading logic relatively easy. Both languages are syntactically rich to allow code that is easy to read and interpret. If business logic requires complex mathematical functions, then a compiled binary language might be needed. The business logic code can be on the client computer, the application server, and the database server. However, the application server is the most common location for business logic.
   - User requests and resource allocation

     Most of this is not affected by the programming language, but tools and fourth generation languages that mask database connection and cursor management might use inefficient mechanisms. When evaluating these tools and environments, check their database connection model and their use of cursors and bind variables.
   - Data management and transactions

     Most of this is not affected by the programming language.
2. When implementing a software component, implement its function and not the functionality associated with other components. Implementing another component's functionality results in sub-optimal designs and implementations. This applies to all components.
3. Do not leave gaps in functionality or have software components under-researched in design, implementation, or testing. In many cases, gaps are not discovered until the application is rolled out or tested at realistic volumes. This is usually a sign of poor

architecture or initial system specification. Data archival and purge modules are most frequently neglected during initial system design, build, and implementation.

4. When implementing procedural logic, implement in a procedural language, such as C, Java, or PL/SQL. When implementing data access (queries) or data changes (DML), use SQL. This rule is specific to the business logic modules of code where procedural code is mixed with data access (nonprocedural SQL) code. There is great temptation to put procedural logic into the SQL access. This tends to result in poor SQL that is resource-intensive. SQL statements with `DECODE` case statements are very often candidates for optimization, as are statements with a large amount of `OR` predicates or set operators, such as `UNION` and `MINUS`.
5. Cache frequently accessed, rarely changing data that is expensive to retrieve on a repeated basis. However, make this cache mechanism easy to use, and ensure that it is indeed cheaper than accessing the data in the original method. This is applicable to all modules where frequently used data values should be cached or stored locally, rather than be repeatedly retrieved from a remote or expensive data store.

   The most common examples of candidates for local caching include the following:

   - Today's date. `SELECT SYSDATE FROM DUAL` can account for over 60% of the workload on a database.
   - The current user name.
   - Repeated application variables and constants, such as tax rates, discounting rates, or location information.
   - Caching data locally can be further extended into building a local data cache into the application server middle tiers. This helps take load off the central database servers. However, care should be taken when constructing local caches so that they do not become so complex that they cease to give a performance gain.
   - Local sequence generation.

   The design implications of using a cache should be considered. For example, if a user is connected at midnight and the date is cached, then the user's date value becomes invalid.
6. Optimize the interfaces between components, and ensure that all components are used in the most scalable configuration. This rule requires minimal explanation and applies to all modules and their interfaces.
7. Use foreign key references. Enforcing referential integrity through an application is expensive. You can maintain a foreign key reference by selecting the column value of the child from the parent and ensuring that it exists. The foreign key constraint enforcement supplied by Oracle—which does not use SQL—is fast, easy to declare, and does not create network traffic.
8. Consider setting up action and module names in the application to use with End to End Application Tracing. This allows greater flexibility in tracing workload problems.

Trends in Application Development

The two biggest challenges in application development today are the increased use of Java to replace compiled C or C++ applications, and increased use of object-oriented techniques, influencing the schema design.

Java provides better portability of code and availability to programmers. However, there are several performance implications associated with Java. Because Java is an interpreted language, it is slower at executing similar logic than compiled languages, such as C. As a result, resource usage of client computers increases. This requires more powerful CPUs to be applied in the client or middle-tier computers and greater care from programmers to produce efficient code.

Because Java is an object-oriented language, it encourages insulation of data access into classes not performing the business logic. As a result, programmers might invoke methods without knowledge of the efficiency of the data access method being used. This tends to result in minimal database access and uses the simplest and crudest interfaces to the database.

With this type of software design, queries do not always include all the `WHERE` predicates to be efficient, and row filtering is performed in the Java program. This is very inefficient. In addition, for DML operations—and especially for `INSERT`s—single `INSERT`s are performed, making use of the array interface impossible. In some cases, this is made more inefficient by procedure calls. More resources are used moving the data to and from the database than in the actual database calls.

In general, it is best to place data access calls next to the business logic to achieve the best overall transaction design.

The acceptance of object-orientation at a programming level has led to the creation of object-oriented databases within the Oracle Server. This has manifested itself in many ways, from storing object structures within `BLOB`s and only using the database effectively as an indexed card file to the use of the Oracle Database object-relational features.

If you adopt an object-oriented approach to schema design, then ensure that you do not lose the flexibility of the relational storage model. In many cases, the object-oriented approach to schema design ends up in a heavily denormalized data structure that requires considerable maintenance and `REF` pointers associated with objects. Often, these designs represent a step backward to the hierarchical and network database designs that were replaced with the relational storage method.

In summary, if you are storing your data in your database for the long-term, and if you anticipate a degree of ad hoc queries or application development on the same schema, then the relational storage method probably gives the best performance and flexibility.

Workload Testing, Modeling, and Implementation

This section describes workload estimation, modeling, implementation, and testing. This section covers the following topics:

Sizing Data

You could experience errors in your sizing estimates when dealing with variable length data if you work with a poor sample set. As data volumes grow, your key lengths could grow considerably, altering your assumptions for column sizes.

When the system becomes operational, it becomes more difficult to predict database growth, especially for indexes. Tables grow over time, and indexes are subject to the individual behavior of the application in terms of key generation, insertion pattern, and deletion of rows. The worst case is where you insert using an ascending key, and then delete most rows from the left-hand side but not all the rows. This leaves gaps and wasted space. If you have index use like this, then ensure that you know how to use the online index rebuild facility.

DBAs should monitor space allocation for each object and look for objects that may grow out of control. A good understanding of the application can highlight objects that may grow rapidly or unpredictably. This is a crucial part of both performance and availability planning for any system. When implementing the production database, the design should attempt to ensure that minimal space management takes place when interactive users are using the application. This applies for all data, temp, and rollback segments.

Estimating Workloads

Considering the number of variables involved, estimation of workloads for capacity planning and testing purposes is extremely difficult. However, designers must specify computers with CPUs, memory, and disk drives, and eventually roll out an application. There are several techniques used for sizing, and each technique has merit. When sizing, it is best to use the following methods to validate your decision-making process and provide supporting documentation.

Extrapolating From a Similar System

This is an entirely empirical approach where an existing system of similar characteristics and known performance is used as a basis system. The specification of this system is then modified by the sizing specialist according to the known differences. This approach has merit in that it correlates with an existing system, but it provides little assistance when dealing with the differences.

This approach is used in nearly all large engineering disciplines when preparing the cost of an engineering project, such as a large building, a ship, a bridge, or an oil rig. If the reference system is an order of magnitude different in size from the anticipated system, then some components may have exceeded their design limits.

Benchmarking

The benchmarking process is both resource and time consuming, and it might not produce the correct results. By simulating an application in early development or prototype form, there is a danger of measuring something that has no resemblance to the actual production system. This sounds strange, but over the many years of benchmarking customer applications with the database development organization, Oracle has yet to see reliable correlation between the benchmark application and the actual production system. This is mainly due to the number of application inefficiencies introduced in the development process.

However, benchmarks have been used successfully to size systems to an acceptable level of accuracy. In particular, benchmarks are very good at determining the actual I/O requirements and testing recovery processes when a system is fully loaded.

Benchmarks by their nature stress all system components to their limits. As the benchmark stresses all components, be prepared to see all errors in application design and implementation manifest themselves while benchmarking. Benchmarks also test database, operating system, and hardware components. Because most

benchmarks are performed in a rush, expect setbacks and problems when a system component fails. Benchmarking is a stressful activity, and it takes considerable experience to get the most out of a benchmarking exercise.

Application Modeling

Modeling the application can range from complex mathematical modeling exercises to the classic simple calculations performed on the back of an envelope. Both methods have merit, with one attempting to be very precise and the other making gross estimates. The downside of both methods is that they do not allow for implementation errors and inefficiencies.

The estimation and sizing process is an imprecise science. However, by investigating the process, some intelligent estimates can be made. The whole estimation process makes no allowances for application inefficiencies introduced by poor SQL, index design, or cursor management. A sizing engineer should build in margin for application inefficiencies. A performance engineer should discover the inefficiencies and make the estimates look realistic. The Oracle performance method describes how to discover the application inefficiencies.

Testing, Debugging, and Validating a Design

The testing process mainly consists of functional and stability testing. At some point in the process, performance testing is performed.

The following list describes some simple rules for performance testing an application. If correctly documented, then this list provides important information for the production application and the capacity planning process after the application has gone live.

- Use the Automatic Database Diagnostic Monitor (ADDM) and SQL Tuning Advisor for design validation
- Test with realistic data volumes and distributions

  All testing must be done with fully populated tables. The test database should contain data representative of the production system in terms of data volume and cardinality between tables. All the production indexes should be built and the schema statistics should be populated correctly.
- Use the correct optimizer mode

  Perform all testing with the optimizer mode that you plan to use in production. All Oracle Database research and development effort is focused on the query optimizer. Therefore, the use of the query optimizer is recommended.
- Test a single user performance

  Test a single user on an idle or lightly-used database for acceptable performance. If a single user cannot achieve acceptable performance under ideal conditions, then multiple users cannot achieve acceptable performance under real conditions.
- Obtain and document plans for all SQL statements

  Obtain an execution plan for each SQL statement. Use this process to verify that the optimizer is obtaining an optimal execution plan, and that the relative cost of the SQL statement is understood in terms of CPU time and physical I/Os. This process assists in identifying the heavy use transactions that require the most tuning and performance work in the future.
- Attempt multiuser testing

This process is difficult to perform accurately, because user workload and profiles might not be fully quantified. However, transactions performing DML statements should be tested to ensure that there are no locking conflicts or serialization problems.

- Test with the correct hardware configuration

  Test with a configuration as close to the production system as possible. Using a realistic system is particularly important for network latencies, I/O subsystem bandwidth, and processor type and speed. Failing to use this approach may result in an incorrect analysis of potential performance problems.

- Measure steady state performance

  When benchmarking, it is important to measure the performance under steady state conditions. Each benchmark run should have a ramp-up phase, where users are connected to the application and gradually start performing work on the application. This process allows for frequently cached data to be initialized into the cache and single execution operations—such as parsing—to be completed before the steady state condition. Likewise, at the end of a benchmark run, there should be a ramp-down period, where resources are freed from the system and users cease work and disconnect.

Deploying New Applications

The following are the key design decisions involved in deploying applications:

- Rollout Strategies
- Performance Checklist

Rollout Strategies

When new applications are rolled out, two strategies are commonly adopted:

- Big Bang approach - all users migrate to the new system at once
- Trickle approach - users slowly migrate from existing systems to the new one

Both approaches have merits and disadvantages. The Big Bang approach relies on reliable testing of the application at the required scale, but has the advantage of minimal data conversion and synchronization with the old system, because it is simply switched off. The Trickle approach allows debugging of scalability issues as the workload increases, but might mean that data must be migrated to and from legacy systems as the transition takes place.

It is difficult to recommend one approach over the other, because each method has associated risks that could lead to system outages as the transition takes place. Certainly, the Trickle approach allows profiling of real users as they are introduced to the new application, and allows the system to be reconfigured while only affecting the migrated users. This approach affects the work of the early adopters, but limits the load on support services. This means that unscheduled outages only affect a small percentage of the user population.

The decision on how to roll out a new application is specific to each business. Any adopted approach has its own unique pressures and stresses. The more testing and knowledge that you derive from the testing process, the more you realize what is best for the rollout.

Performance Checklist

To assist in the rollout, build a list of tasks that increase the chance of optimal performance in production and enable rapid debugging of the application. Do the following:

1. When you create the control file for the production database, allow for growth by setting `MAXINSTANCES`, `MAXDATAFILES`, `MAXLOGFILES`, `MAXLOGMEMBERS`, and `MAXLOGHISTORY` to values higher than what you anticipate for the rollout. This technique results in more disk space usage and larger control files, but saves time later should these need extension in an emergency.
2. Set block size to the value used to develop the application. Export the schema statistics from the development or test environment to the production database if the testing was done on representative data volumes and the current SQL execution plans are correct.
3. Set the minimal number of initialization parameters. Ideally, most other parameters should be left at default. If there is more tuning to perform, then this appears when the system is under load.
4. Be prepared to manage block contention by setting storage options of database objects. Tables and indexes that experience high `INSERT`/`UPDATE`/`DELETE` rates should be created with automatic segment space management. To avoid contention of rollback segments, use automatic undo management.
5. All SQL statements should be verified to be optimal and their resource usage understood.
6. Validate that middleware and programs that connect to the database are efficient in their connection management and do not logon or logoff repeatedly.
7. Validate that the SQL statements use cursors efficiently. The database should parse each SQL statement once and then execute it multiple times. The most common reason this does not happen is because bind variables are not used properly and `WHERE` clause predicates are sent as string literals. If you use precompilers to develop the application, then make sure to reset the parameters `MAXOPENCURSORS`, `HOLD_CURSOR`, and `RELEASE_CURSOR` from the default values before precompiling the application.
8. Validate that all schema objects have been correctly migrated from the development environment to the production database. This includes tables, indexes, sequences, triggers, packages, procedures, functions, Java objects, synonyms, grants, and views. Ensure that any modifications made in testing are made to the production system.
9. As soon as the system is rolled out, establish a baseline set of statistics from the database and operating system. This first set of statistics validates or corrects any assumptions made in the design and rollout process.
10. Start anticipating the first bottleneck (which is inevitable) and follow the Oracle performance method to make performance improvement.

3
Performance Improvement Methods

This chapter discusses Oracle Database improvement methods and contains the following sections:

The Oracle Performance Improvement Method

Oracle performance methodology helps you to identify performance problems in an Oracle database. This involves identifying bottlenecks and fixing them. It is recommended that changes be made to a system only after you have confirmed that there is a bottleneck.

Performance improvement, by its nature, is iterative. For this reason, removing the first bottleneck might not lead to performance improvement immediately, because another bottleneck might be revealed. Also, in some cases, if serialization points move to a more inefficient sharing mechanism, then performance could degrade. With experience, and by following a rigorous method of bottleneck elimination, applications can be debugged and made scalable.

Performance problems generally result from either a lack of throughput, unacceptable user/job response time, or both. The problem might be localized between application modules, or it might be for the entire system.

Before looking at any database or operating system statistics, it is crucial to get feedback from the most important components of the system: the users of the system and the people ultimately paying for the application. Typical user feedback includes statements like the following:

- "The online performance is so bad that it prevents my staff from doing their jobs."
- "The billing run takes too long."
- "When I experience high amounts of Web traffic, the response time becomes unacceptable, and I am losing customers."
- "I am currently performing 5000 trades a day, and the system is maxed out. Next month, we roll out to all our users, and the number of trades is expected to quadruple."

From candid feedback, it is easy to set critical success factors for any performance work. Determining the performance targets and the performance engineer's exit criteria make managing the performance process much simpler and more successful at all levels. These critical success factors are better defined in terms of real business goals rather than system statistics.

Some real business goals for these typical user statements might be:

- "The billing run must process 1,000,000 accounts in a three-hour window."
- "At a peak period on a Web site, the response time must not exceed five seconds for a page refresh."
- "The system must be able to process 25,000 trades in an eight-hour window."

The ultimate measure of success is the user's perception of system performance. The performance engineer's role is to eliminate any bottlenecks that degrade performance. These bottlenecks could be caused by inefficient use of limited shared resources or by abuse of shared resources, causing serialization. Because all shared resources are limited, the goal of a performance engineer is to maximize the number of business operations with efficient use of shared resources. At a very high level, the entire database server can be seen as a shared resource. Conversely, at a low level, a single CPU or disk can be seen as shared resources.

You can apply the Oracle performance improvement method until performance goals are met or deemed impossible. This process is highly iterative. Inevitably, some investigations may have little or no impact on database performance. Time and experience are necessary to develop the skills to accurately and quickly pinpoint critical bottlenecks. However, prior experience can sometimes work against the experienced engineer who neglects to use the data and statistics available. This type of behavior encourages database tuning by myth and folklore. This is a very risky, expensive, and unlikely to succeed method of database tuning.

The Automatic Database Diagnostic Monitor (ADDM) implements parts of the performance improvement method and analyzes statistics to provide automatic diagnosis of major performance issues. Using ADDM can significantly shorten the time required to improve the performance of a system.

Systems are so different and complex that hard and fast rules for performance analysis are impossible. In essence, the Oracle performance improvement method defines a way of working, but not a definitive set of rules. With bottleneck detection, the only rule is that there are no rules! The best performance engineers use the data provided and think laterally to determine performance problems.

Steps in the Oracle Performance Improvement Method

1. Perform the following initial standard checks:
    a. Get candid feedback from users. Determine the performance project's scope and subsequent performance goals, and performance goals for the future. This process is key in future capacity planning.
    b. Get a full set of operating system, database, and application statistics from the system when the performance is both good and bad. If these are not available, then get whatever is available. Missing statistics are analogous to missing evidence at a crime scene: They make detectives work harder and it is more time-consuming.
    c. Sanity-check the operating systems of all computers involved with user performance. By sanity-checking the operating system, you look for hardware or operating system resources that are fully utilized. List any over-used resources as symptoms for analysis later. In addition, check that all hardware shows no errors or diagnostics.
2. Check for the top ten most common mistakes with Oracle Database, and determine if any of these are likely to be the problem. List these as symptoms for later analysis. These are included because they represent the most likely problems. ADDM automatically detects and reports nine of these top ten issues.
3. Build a conceptual model of what is happening on the system using the symptoms as clues to understand what caused the performance problems. See "A Sample Decision Process for Performance Conceptual Modeling".

4. Propose a series of remedy actions and the anticipated behavior to the system, then apply them in the order that can benefit the application the most. ADDM produces recommendations each with an expected benefit. A golden rule in performance work is that you only change one thing at a time and then measure the differences. Unfortunately, system downtime requirements might prohibit such a rigorous investigation method. If multiple changes are applied at the same time, then try to ensure that they are isolated so that the effects of each change can be independently validated.
5. Validate that the changes made have had the desired effect, and see if the user's perception of performance has improved. Otherwise, look for more bottlenecks, and continue refining the conceptual model until your understanding of the application becomes more accurate.
6. Repeat the last three steps until performance goals are met or become impossible due to other constraints.

This method identifies the biggest bottleneck and uses an objective approach to performance improvement. The focus is on making large performance improvements by increasing application efficiency and eliminating resource shortages and bottlenecks. In this process, it is anticipated that minimal (less than 10%) performance gains are made from instance tuning, and large gains (100% +) are made from isolating application inefficiencies.

A Sample Decision Process for Performance Conceptual Modeling

Conceptual modeling is almost deterministic. However, as you gain experience in performance tuning, you begin to appreciate that no real rules exist. A flexible heads-up approach is required to interpret statistics and make good decisions.

For a quick and easy approach to performance tuning, use ADDM. ADDM automatically monitors your Oracle system and provides recommendations for solving performance problems should problems occur. For example, suppose a DBA receives a call from a user complaining that the system is slow. The DBA simply examines the latest ADDM report to see which of the recommendations should be implemented to solve the problem.

The following steps illustrate how a performance engineer might look for bottlenecks without using automatic diagnostic features. These steps are only intended as a guideline for the manual process. With experience, performance engineers add to the steps involved. This analysis assumes that statistics for both the operating system and the database have been gathered.

1. Is the response time/batch run time acceptable for a single user on an empty or lightly loaded computer?

   If it is not acceptable, then the application is probably not coded or designed optimally, and it will never be acceptable in a multiple user situation when system resources are shared. In this case, get application internal statistics, and get SQL Trace and SQL plan information. Work with developers to investigate problems in data, index, transaction SQL design, and potential deferral of work to batch and background processing.

2. Is all the CPU being utilized?

   If the kernel utilization is over 40%, then investigate the operating system for network transfers, paging, swapping, or process thrashing. Continue to check CPU utilization in user space to verify if there are any non-database jobs consuming CPU on the system limiting the amount of shared CPU resources, such as backups, file transforms, print queues, and so on. After determining that the database is using most of the CPU, investigate the top SQL by CPU utilization. These statements form the basis of all future

analysis. Check the SQL and the transactions submitting the SQL for optimal execution. Oracle Database provides CPU statistics in `V$SQL` and `V$SQLSTATS`.

> **See Also:**
>
> *Oracle Database Reference* for more information about `V$SQL` and `V$SQLSTATS`

If the application is optimal and no inefficiencies exist in the SQL execution, then consider rescheduling some work to off-peak hours or using a bigger computer.

3. At this point, the system performance is unsatisfactory, yet the CPU resources are not fully utilized.

   In this case, you have serialization and unscalable behavior within the server. Get the `WAIT_EVENTS` statistics from the server, and determine the biggest serialization point. If there are no serialization points, then the problem is most likely outside the database, and this should be the focus of investigation. Elimination of `WAIT_EVENTS` involves modifying application SQL and tuning database parameters. This process is very iterative and requires the ability to drill down on the `WAIT_EVENTS` systematically to eliminate serialization points.

Top Ten Mistakes Found in Oracle Systems

This section lists the most common mistakes found in Oracle databases. By following the Oracle performance improvement methodology, you should be able to avoid these mistakes altogether. If you find these mistakes in your system, then re-engineer the application where the performance effort is worthwhile.

1. Bad connection management

   The application connects and disconnects for each database interaction. This problem is common with stateless middleware in application servers. It has over two orders of magnitude impact on performance, and is totally unscalable.

2. Bad use of cursors and the shared pool

   Not using cursors results in repeated parses. If bind variables are not used, then there is hard parsing of all SQL statements. This has an order of magnitude impact in performance, and it is totally unscalable. Use cursors with bind variables that open the cursor and execute it many times. Be suspicious of applications generating dynamic SQL.

3. Bad SQL

   Bad SQL is SQL that uses more resources than appropriate for the application requirement. This can be a decision support systems (DSS) query that runs for more than 24 hours, or a query from an online application that takes more than a minute. You should investigate SQL that consumes significant system resources for potential improvement. ADDM identifies high load SQL. SQL Tuning Advisor can provide recommendations for improvement.

4. Use of nonstandard initialization parameters

   These might have been implemented based on poor advice or incorrect assumptions. Most databases provide acceptable performance using only the set of basic parameters. In particular, parameters associated with `SPIN_COUNT` on

latches and undocumented optimizer features can cause a great deal of problems that can require considerable investigation.

Likewise, optimizer parameters set in the initialization parameter file can override proven optimal execution plans. For these reasons, schemas, schema statistics, and optimizer settings should be managed as a group to ensure consistency of performance.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about initialization parameters and database creation
> - *Oracle Database Reference* for details on initialization parameters

5. Getting database I/O wrong

   Many sites lay out their databases poorly over the available disks. Other sites specify the number of disks incorrectly, because they configure disks by disk space and not I/O bandwidth.

6. Online redo log setup problems

   Many sites run with too few online redo log files and files that are too small. Small redo log files cause system checkpoints to continuously put a high load on the buffer cache and I/O system. If too few redo log files exist, then the archive cannot keep up, and the database must wait for the archiver to catch up.

7. Serialization of data blocks in the buffer cache due to lack of free lists, free list groups, transaction slots (`INITRANS`), or shortage of rollback segments.

   This is particularly common on `INSERT`-heavy applications, in applications that have raised the block size above 8K, or in applications with large numbers of active users and few rollback segments. Use automatic segment-space management (ASSM) and automatic undo management to solve this problem.

8. Long full table scans

   Long full table scans for high-volume or interactive online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Long table scans, by nature, are I/O intensive and unscalable.

9. High amounts of recursive (`SYS`) SQL

   Large amounts of recursive SQL executed by `SYS` could indicate space management activities, such as extent allocations, taking place. This is unscalable and impacts user response time. Use locally managed tablespaces to reduce recursive SQL due to extent allocation. Recursive SQL executed under another user ID is probably SQL and PL/SQL, and this is not a problem.

10. Deployment and migration errors

    In many cases, an application uses too many resources because the schema owning the tables has not been successfully migrated from the development environment or from an older implementation. Examples of this are missing indexes or incorrect statistics. These errors can lead to sub-optimal execution plans and poor interactive user performance. When migrating applications of known performance, export the schema statistics to maintain plan stability using the `DBMS_STATS` package.

Although these errors are not directly detected by ADDM, ADDM highlights the resulting high load SQL.

Emergency Performance Methods

This section provides techniques for dealing with performance emergencies. You presumably have a methodology for establishing and improving application performance. However, in an emergency situation, a component of the system has changed to transform it from a reliable, predictable system to one that is unpredictable and not satisfying user requests.

In this case, the performance engineer must rapidly determine what has changed and take appropriate actions to resume normal service as quickly as possible. In many cases, it is necessary to take immediate action, and a rigorous performance improvement project is unrealistic.

After addressing the immediate performance problem, the performance engineer must collect sufficient debugging information either to get better clarity on the performance problem or to at least ensure that it does not happen again.

The method for debugging emergency performance problems is the same as the method described in the performance improvement method earlier in this book. However, shortcuts are taken in various stages because of the timely nature of the problem. Keeping detailed notes and records of facts found as the debugging process progresses is essential for later analysis and justification of any remedial actions. This is analogous to a doctor keeping good patient notes for future reference.

Steps in the Emergency Performance Method

The Emergency Performance Method is as follows:

1. Survey the performance problem and collect the symptoms of the performance problem. This process should include the following:
   - User feedback on how the system is underperforming. Is the problem throughput or response time?
   - Ask the question, "What has changed since we last had good performance?" This answer can give clues to the problem. However, getting unbiased answers in an escalated situation can be difficult. Try to locate some reference points, such as collected statistics or log files, that were taken before and after the problem.
   - Use automatic tuning features to diagnose and monitor the problem. In addition, you can use Oracle Enterprise Manager Cloud Control (Cloud Control) performance features to identify top SQL and sessions.
2. Sanity-check the hardware utilization of all components of the application system. Check where the highest CPU utilization is, and check the disk, memory usage, and network performance on all the system components. This quick process identifies which tier is causing the problem. If the problem is in the application, then shift analysis to application debugging. Otherwise, move on to database server analysis.
3. Determine if the database server is constrained on CPU or if it is spending time waiting on wait events. If the database server is CPU-constrained, then investigate the following:

- Sessions that are consuming large amounts of CPU at the operating system level and database; check `V$SESS_TIME_MODEL` for database CPU usage
- Sessions or statements that perform many buffer gets at the database level; check `V$SESSTAT` and `V$SQLSTATS`
- Execution plan changes causing sub-optimal SQL execution; these can be difficult to locate
- Incorrect setting of initialization parameters
- Algorithmic issues caused by code changes or upgrades of all components

If the database sessions are waiting on events, then follow the wait events listed in `V$SESSION_WAIT` to determine what is causing serialization. The `V$ACTIVE_SESSION_HISTORY` view contains a sampled history of session activity which you can use to perform diagnosis even after an incident has ended and the system has returned to normal operation. In cases of massive contention for the library cache, it might not be possible to logon or submit SQL to the database. In this case, use historical data to determine why there is suddenly contention on this latch. If most waits are for I/O, then examine `V$ACTIVE_SESSION_HISTORY` to determine the SQL being run by the sessions that are performing all of the inputs and outputs.

4. Apply emergency action to stabilize the system. This could involve actions that take parts of the application off-line or restrict the workload that can be applied to the system. It could also involve a system restart or the termination of job in process. These naturally have service level implications.
5. Validate that the system is stable. Having made changes and restrictions to the system, validate that the system is now stable, and collect a reference set of statistics for the database. Now follow the rigorous performance method described earlier in this book to bring back all functionality and users to the system. This process may require significant application re-engineering before it is complete.

4

Configuring a Database for Performance

This chapter contains an overview of the Oracle methodology for configuring a database for performance. Although performance modifications can be made to Oracle Database on an ongoing basis, significant benefits can be gained by proper initial configuration of the database.

This chapter contains the following sections:

Performance Considerations for Initial Instance Configuration

The initial database instance configuration options that have important performance impact on the database are:

- Initialization Parameters
- Undo Space
- Redo Log Files
- Tablespaces

> **Note:**
>
> If you use the Database Configuration Assistant (DBCA) to create a database, then the supplied seed database includes the necessary basic initialization parameters and meet the performance recommendations that are mentioned in this document.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn how to create a database with the Database Configuration Assistant
> - *Oracle Database Administrator's Guide* to learn how to create a database with a SQL statement

Initialization Parameters

A running Oracle database instance is configured using initialization parameters, which are set in the initialization parameter file. These parameters influence the behavior of the running instance, including influencing performance. In general, a very simple initialization file with few relevant settings covers most situations, and the initialization file should not be the first place you expect to do performance tuning, except for the few parameters described in the following table.

The following table describes the parameters necessary in a minimal initialization file. Although these parameters are necessary, they have no performance impact.

Table 4-1 Necessary Initialization Parameters Without Performance Impact

Parameter	Description
DB_NAME	Name of the database. This should match the ORACLE_SID environment variable.
DB_DOMAIN	Location of the database in Internet dot notation.
OPEN_CURSORS	Limit on the maximum number of cursors (active SQL statements) for each session. The setting is application-dependent; 500 is recommended.
CONTROL_FILES	Set to contain at least two files on different disk drives to prevent failures from control file loss.
DB_FILES	Set to the maximum number of files that can assigned to the database.

The following table includes the most important parameters to set with performance implications:

Table 4-2 Important Initialization Parameters With Performance Impact

Parameter	Description
COMPATIBLE	Specifies the release with which the Oracle database must maintain compatibility. It lets you take advantage of the maintenance improvements of a new release immediately in your production systems without testing the new functionality in your environment. If your application was designed for a specific release of Oracle Database, and you are actually installing a later release, then you might want to set this parameter to the version of the previous release.
DB_BLOCK_SIZE	Sets the size of the Oracle database blocks stored in the database files and cached in the SGA. The range of values depends on the operating system, but it is typically 8192 for transaction processing systems and higher values for database warehouse systems.
SGA_TARGET	Specifies the total size of all SGA components. If SGA_TARGET is specified, then the buffer cache (DB_CACHE_SIZE), Java pool (JAVA_POOL_SIZE), large pool (LARGE_POOL_SIZE), and shared pool (SHARED_POOL_SIZE) memory pools are automatically sized.

Table 4-2 (Cont.) Important Initialization Parameters With Performance Impact

Parameter	Description
PGA_AGGREGATE_TARGET	Specifies the target aggregate PGA memory available to all server processes attached to the instance.
PROCESSES	Sets the maximum number of processes that can be started by that instance. This is the most important primary parameter to set, because many other parameter values are deduced from this.
SESSIONS	This is set by default from the value of processes. However, if you are using the shared server, then the deduced value is likely to be insufficient.
UNDO_MANAGEMENT	Specifies the undo space management mode used by the database. The default is AUTO. If unspecified, the database uses AUTO.
UNDO_TABLESPACE	Specifies the undo tablespace to be used when an instance starts.

See Also:

The following guides for more information about these initialization parameters:

- *Oracle Database Administrator's Guide*
- *Oracle Database Reference*

Undo Space

The database uses undo space to store data used for read consistency, recovery, and rollback statements. This data exists in one or more undo tablespaces. If you use the Database Configuration Assistant (DBCA) to create a database, then the undo tablespace is created automatically. To manually create an undo tablespace, add the UNDO TABLESPACE clause to the CREATE DATABASE statement.

To automate the management of undo data, Oracle Database uses **automatic undo management**, which transparently creates and manages undo segments.To enable automatic undo management, set the UNDO_MANAGEMENT initialization parameter to AUTO (the default setting). If unspecified, then the UNDO_MANAGEMENT initialization parameter uses the AUTO setting. Oracle strongly recommends using automatic undo management because it significantly simplifies database management and eliminates the need for any manual tuning of undo (rollback) segments. Manual undo management using rollback segments is supported for backward compatibility.

The V$UNDOSTAT view contains statistics for monitoring and tuning undo space. Using this view, you can better estimate the amount of undo space required for the current workload. Oracle Database also uses this information to help tune undo usage. The V$ROLLSTAT view contains information about the behavior of the undo segments in the undo tablespace.

> **See Also:**
>
> - *Oracle Database 2 Day DBA* and Oracle Enterprise Manager Cloud Control (Cloud Control) online help to learn about the Undo Management Advisor
> - *Oracle Database Administrator's Guide* for information about managing undo space using automatic undo management
> - *Oracle Database Reference* for more information about the `V$ROLLSTAT` view
> - *Oracle Database Reference* for more information about the `V$UNDOSTAT` view

Redo Log Files

The size of the redo log files can influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance. Undersized log files increase checkpoint activity and reduce performance.

Although the size of the redo log files does not affect LGWR performance, it can affect DBWR and checkpoint behavior. Checkpoint frequency is affected by several factors, including log file size and the setting of the `FAST_START_MTTR_TARGET` initialization parameter. If the `FAST_START_MTTR_TARGET` parameter is set to limit the instance recovery time, Oracle Database automatically tries to checkpoint as frequently as necessary. Under this condition, the size of the log files should be large enough to avoid additional checkpointing due to under sized log files. The optimal size can be obtained by querying the `OPTIMAL_LOGFILE_SIZE` column from the `V$INSTANCE_RECOVERY` view. You can also obtain sizing advice on the **Redo Log Groups** page of Oracle Enterprise Manager Cloud Control (Cloud Control).

It may not always be possible to provide a specific size recommendation for redo log files, but redo log files in the range of 100 MB to a few gigabytes are considered reasonable. Size online redo log files according to the amount of redo your system generates. A rough guide is to switch log files at most once every 20 minutes.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about managing the online redo log

Tablespaces

If you use the Database Configuration Assistant (DBCA) to create a database, then the seed database automatically includes the necessary tablespaces. If you choose not to use DBCA, then you must create extra tablespaces after creating the database.

All databases should have several tablespaces in addition to the `SYSTEM` and `SYSAUX` tablespaces. These additional tablespaces include:

- A temporary tablespace, which is used for operations such as sorting
- An undo tablespace to contain information for read consistency, recovery, and undo statements
- At least one tablespace for application use (in most cases, applications require several tablespaces)

For extremely large tablespaces with many data files, you can run multiple `ALTER TABLESPACE ... ADD DATAFILE` statements in parallel. During tablespace creation, the data files that make up the tablespace are initialized with special empty block images. Temporary files are not initialized.

Oracle Database does this to ensure that it can write all data files in their entirety, but this can obviously be a lengthy process if done serially. Therefore, run multiple `CREATE TABLESPACE` statements concurrently to speed up tablespace creation. For permanent tables, the choice between local and global extent management on tablespace creation can greatly affect performance. For any permanent tablespace that has moderate to large insert, modify, or delete operations compared to reads, choose local extent management.

Permanent Tablespaces - Automatic Segment-Space Management

For permanent tablespaces, Oracle recommends using automatic segment-space management. Such tablespaces, often referred to as bitmap tablespaces, are locally managed tablespaces with bitmap segment space management.

> **See Also:**
>
> - *Oracle Database Concepts* for a discussion of free space management
> - *Oracle Database Administrator's Guide* for more information on creating and using automatic segment-space management for tablespaces

Temporary Tablespaces

Properly configuring the temporary tablespace helps optimize disk sort performance. Temporary tablespaces can be dictionary-managed or locally managed. Oracle recommends the use of locally managed temporary tablespaces with a `UNIFORM` extent size of 1 `MB`.

You should monitor temporary tablespace activity to check how many extents the database allocates for the temporary segment. If an application extensively uses temporary tables, as in a situation when many users are concurrently using temporary tables, then the extent size could be set smaller, such as 256K, because every usage requires at least one extent. The `EXTENT MANAGEMENT LOCAL` clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The default for `SIZE` is `1M`.

See Also:

- *Oracle Database Administrator's Guide* for more information on managing temporary tablespaces
- *Oracle Database Concepts* for more information on temporary tablespaces
- *Oracle Database SQL Language Reference* for more information on using the `CREATE` and `ALTER TABLESPACE` statements with the `TEMPORARY` clause

Creating and Maintaining Tables for Optimal Performance

When installing applications, an initial step is to create all necessary tables and indexes. When you create a segment, such as a table, the database allocates space for the data. If subsequent database operations cause the data volume to increase and exceed the space allocated, then Oracle Database extends the segment.

When creating tables and indexes, note the following:

- Specify automatic segment-space management for tablespaces

  In this way Oracle Database automatically manages segment space for best performance.

- Set storage options carefully

  Applications should carefully set storage options for the intended use of the table or index. This includes setting the value for `PCTFREE`. Note that using automatic segment-space management eliminates the necessity of specifying `PCTUSED`.

Note:

Use of free lists is not recommended. To use automatic segment-space management, create locally managed tablespaces, with the segment space management clause set to `AUTO`.

Table Compression

You can store heap-organized tables in a compressed format that is transparent for any kind of application. Compressed data in a database block is self-contained, which means that all information needed to re-create the uncompressed data in a block is available within the block. A block is also compressed in the buffer cache. Table compression not only reduces the disk storage but also the memory usage, specifically the buffer cache requirements. Performance improvements are accomplished by reducing the amount of necessary I/O operations for accessing a table and by increasing the probability of buffer cache hits.

Oracle Database has an advanced compression option that enables you to boost the performance of any type of application workload—including data warehousing and OLTP applications—while reducing the disk storage that is required by the database.

You can use the advanced compression feature for all types of data, including structured data, unstructured data, backup data, and network data.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about using table compression

Estimating the Compression Factor

Table compression works by eliminating column value repetitions within individual blocks. Duplicate values in all the rows and columns in a block are stored once at the beginning of the block, in what is called a symbol table for that block. All occurrences of such values are replaced with a short reference to the symbol table. The compression is higher in blocks that have more repeated values.

Before compressing large tables you should estimate the expected compression factor. The compression factor is defined as the number of blocks necessary to store the information in an uncompressed form divided by the number of blocks necessary for a compressed storage. The compression factor can be estimated by sampling a small number of representative data blocks of the table to be compressed and comparing the average number of records for each block for the uncompressed and compressed case. Experience shows that approximately 1000 data blocks provides a very accurate estimation of the compression factor. Note that the more blocks you are sampling, the more accurate the results become.

Tuning to Achieve a Better Compression Ratio

Oracle Database achieves a good compression factor in many cases with no special tuning. As a DBA or application developer, you can try to tune the compression factor by reorganizing the records when the compression takes place. Tuning can improve the compression factor slightly in some cases and substantially in other cases.

To improve the compression factor you must increase the likelihood of value repetitions within a data block. The achievable compression factor depends on the cardinality of a specific column or column pairs (representing the likelihood of column value repetitions) and on the average row length of those columns. Table compression not only compresses duplicate values of a single column but tries to use multi-column value pairs whenever possible. Without a detailed understanding of the data distribution it is very difficult to predict the most optimal order.

Using Attribute-Clustered Tables

An attribute-clustered table is a heap-organized table that stores data in close proximity on disk based on user-specified clustering directives. The directives determine if the data stored in a table is ordered based on specified columns, or on a special algorithm that permits multicolumn I/O reduction. Attribute clustering is only available for bulk insert operations—such as the `INSERT/*+APPEND*/` or `ALTER TABLE ... MOVE PARTITION` commands—and is ignored for conventional DML.

By reducing physical I/O in conjunction with zone maps, using attribute-clustered tables can significant reduce the I/O costs of table scans. Furthermore, it can also improve data compression because data can be more easily compressed when the same values are closer to each other on disk.

> **See Also:**
>
> - *Oracle Database Concepts* for information about attribute-clustered tables
> - *Oracle Database Data Warehousing Guide* for information about using attribute-clustered tables

Reclaiming Unused Space

Over time, it is common for segment space to become fragmented or for a segment to acquire a lot of free space as the result of update and delete operations. The resulting sparsely populated objects can suffer performance degradation during queries and DML operations. If an object does have space available for reclamation, then you can compact and shrink segments or deallocate unused space at the end of a segment.

Oracle Database provides a Segment Advisor that provides advice on whether an object has space available for reclamation based on the level of space fragmentation within an object.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for a discussion on managing space for schema objects and the Segment Advisor

Indexing Data

The most efficient time to create indexes is after data has been loaded. In this way, space management becomes simpler, and no index maintenance takes place for each row inserted. SQL*Loader automatically uses this technique, but if you are using other methods to do initial data load, then you may need to create indexes manually. Additionally, you can perform index creation in parallel using the `PARALLEL` clause of the `CREATE INDEX` statement. However, SQL*Loader is not able to parallelize index creation, so you must manually create indexes in parallel after loading data.

> **See Also:**
>
> *Oracle Database Utilities* for information about SQL*Loader

Specifying Memory for Sorting Data

During index creation on tables that contain data, the data must be sorted. This sorting is done in the fastest possible way, if all available memory is used for sorting. Oracle recommends that you enable automatic sizing of SQL working areas by setting the `PGA_AGGREGATE_TARGET` initialization parameter.

> **See Also:**
>
> - "Tuning the Program Global Area " for information about PGA memory management
> - *Oracle Database Reference* for information about the `PGA_AGGREGATE_TARGET` initialization parameter

Performance Considerations for Shared Servers

Using shared servers reduces the number of processes and the amount of memory consumed on the database host. Shared servers are beneficial for databases where there are many OLTP users performing intermittent transactions.

Using shared servers rather than dedicated servers is also generally better for systems that have a high connection rate to the database. With shared servers, when a connect request is received, a dispatcher is available to handle concurrent connection requests. With dedicated servers, however, a connection-specific dedicated server is sequentially initialized for each connection request.

Performance of certain database features can improve when a shared server architecture is used, and performance of certain database features can degrade slightly when a shared server architecture is used. For example, a session can be prevented from migrating to another shared server while parallel execution is active.

A session can remain nonmigratable even after a request from the client has been processed, because not all the user information has been stored in the UGA. If a server were to process the request from the client, then the part of the user state that was not stored in the UGA would be inaccessible. To avoid this situation, individual shared servers often need to remain bound to a user session.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn how to manage shared servers
> - *Oracle Database Net Services Administrator's Guide* to learn how to configure dispatchers for shared servers

When using some features, you may need to configure more shared servers, because some servers might be bound to sessions for an excessive amount of time.

This section discusses how to reduce contention for processes used by Oracle Database architecture:

- Identifying and Reducing Contention Using the Dispatcher-Specific Views
- Identifying Contention for Shared Servers

Identifying and Reducing Contention Using the Dispatcher-Specific Views

The following views provide dispatcher performance statistics:

- V$DISPATCHER: general information about dispatcher processes
- V$DISPATCHER_RATE: dispatcher processing statistics

The V$DISPATCHER_RATE view contains current, average, and maximum dispatcher statistics for several categories. Statistics with the prefix CUR_ are statistics for the current sample. Statistics with the prefix AVG_ are the average values for the statistics after the collection period began. Statistics with the prefix MAX_ are the maximum values for these categories after statistics collection began.

To assess dispatcher performance, query the V$DISPATCHER_RATE view and compare the current values with the maximums. If your present system throughput provides adequate response time and current values from this view are near the average and less than the maximum, then you likely have an optimally tuned shared server environment.

If the current and average rates are significantly less than the maximums, then consider reducing the number of dispatchers. Conversely, if current and average rates are close to the maximums, then you might need to add more dispatchers. A general rule is to examine V$DISPATCHER_RATE statistics during both light and heavy system use periods. After identifying your shared server load patterns, adjust your parameters accordingly.

If necessary, you can also mimic processing loads by running system stress tests and periodically polling V$DISPATCHER_RATE statistics. Proper interpretation of these statistics varies from platform to platform. Different types of applications also can cause significant variations on the statistical values recorded in V$DISPATCHER_RATE.

See Also:

- *Oracle Database Reference* for detailed information about the V$DISPATCHER and V$DISPATCHER_RATE views

Reducing Contention for Dispatcher Processes

To reduce contention, consider the following points:

- Adding dispatcher processes

  The total number of dispatcher processes is limited by the value of the initialization parameter MAX_DISPATCHERS. You might need to increase this value before adding dispatcher processes.

- Enabling connection pooling

  When system load increases and dispatcher throughput is maximized, it is not necessarily a good idea to immediately add more dispatchers. Instead, consider configuring the dispatcher to support more users with connection pooling.

- Enabling Session Multiplexing

  Multiplexing is used by a connection manager process to establish and maintain network sessions from multiple users to individual dispatchers. For example, several user processes can connect to one dispatcher by way of a single connection from a connection manager process. Session multiplexing is beneficial

because it maximizes use of the dispatcher process connections. Multiplexing is also useful for multiplexing database link sessions between dispatchers.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn how to configure dispatcher processes
> - *Oracle Database Net Services Administrator's Guide* to learn how to configure connection pooling
> - *Oracle Database Reference* to learn about the `DISPATCHERS` and `MAX_DISPATCHERS` initialization parameters

Identifying Contention for Shared Servers

Steadily increasing wait times in the requests queue indicate contention for shared servers. To examine wait time data, use the dynamic performance view `V$QUEUE`. This view contains statistics showing request queue activity for shared servers. By default, this view is available only to the user `SYS` and to other users with `SELECT ANY TABLE` system privilege, such as `SYSTEM`. Table 4-3 lists the columns showing the wait times for requests and the number of requests in the queue.

Table 4-3 Wait Time and Request Columns in V$QUEUE

Column	Description
`WAIT`	Displays the total waiting time, in hundredths of a second, for all requests that have ever been in the queue
`TOTALQ`	Displays the total number of requests that have ever been in the queue

Monitor these statistics occasionally while your application is running by issuing the following SQL statement:

```
SELECT DECODE(TOTALQ, 0, 'No Requests',
   WAIT/TOTALQ || ' HUNDREDTHS OF SECONDS') "AVERAGE WAIT TIME PER REQUESTS"
  FROM V$QUEUE
 WHERE TYPE = 'COMMON';
```

This query returns the results of a calculation that show the following:

```
AVERAGE WAIT TIME PER REQUEST
-----------------------------
.090909 HUNDREDTHS OF SECONDS
```

From the result, you can tell that a request waits an average of 0.09 hundredths of a second in the queue before processing.

You can also determine how many shared servers are currently running by issuing the following query:

```
SELECT COUNT(*) "Shared Server Processes"
  FROM V$SHARED_SERVER
 WHERE STATUS != 'QUIT';
```

The result of this query could look like the following:

```
Shared Server Processes
----------------------
10
```

If you detect resource contention with shared servers, then first ensure that this is not a memory contention issue by examining the shared pool and the large pool. If performance remains poor, then you might want to create more resources to reduce shared server process contention. You can do this by modifying the optional server process initialization parameters:

- `MAX_DISPATCHERS`
- `MAX_SHARED_SERVERS`
- `DISPATCHERS`
- `SHARED_SERVERS`

> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn how to set the shared server process initialization parameters

Improved Client Connection Performance Due to Prespawned Processes

Oracle Database prespawns pools of server processes when dedicated broker connection mode is enabled or threaded execution mode is enabled. In this case, whenever a client requests for a database connection, it gets a dedicated connection to an existing server process from the process pools, thus improving the efficiency of client connections.

The `V$PROCESS_POOL` view shows information about these server process pools, and you can manage these pools using the `DBMS_PROCESS` package.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about managing prespawned processes in Oracle Database
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_PROCESS` package

Part II

Diagnosing and Tuning Database Performance

This part contains the following chapters:

5

Measuring Database Performance

This chapter describes how to measure the performance of Oracle Database using database statistics.

This chapter contains the following topics:

- About Database Statistics
- Interpreting Database Statistics

About Database Statistics

Database statistics provide information about the type of database load and the resources being used by the database. To effectively measure database performance, statistics must be available.

Oracle Database generates many types of cumulative statistics for the system, sessions, segments, services, and individual SQL statements. Cumulative values for statistics are generally accessible using dynamic performance views, or `V$` views. When analyzing database performance in any of these scopes, look at the change in statistics (delta value) over the period you are interested in. Specifically, focus on the difference between the cumulative values of a statistic at the start and the end of the period.

This section describes some of the more important database statistics that are used to measure the performance of Oracle Database:

- Time Model Statistics
- Active Session History Statistics
- Wait Events Statistics
- Session and System Statistics

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for information about optimizer statistics

Time Model Statistics

Time model statistics use time to identify quantitative effects about specific actions performed on the database, such as logon operations and parsing. The most important time model statistic is database time, or DB time. This statistic represents the total time spent in database calls for foreground sessions and is an indicator of the total instance workload. DB time is measured cumulatively from the time of instance startup and is calculated by aggregating the CPU and wait times of all foreground sessions not waiting on idle wait events (non-idle user sessions).

> **Note:**
>
> Because DB time is calculated by combining the times from all non-idle user foreground sessions, it is possible that the DB time can exceed the actual time elapsed after the instance started. For example, an instance that has been running for 30 minutes could have four active user sessions whose cumulative DB time is approximately 120 minutes.

When tuning an Oracle database, each component has its own set of statistics. To look at the system as a whole, it is necessary to have a common scale for comparisons. Many Oracle Database advisors and reports thus describe statistics in terms of time.

Ultimately, the objective in tuning an Oracle database is to reduce the time that users spend in performing an action on the database, or to simply reduce `DB time`. Time model statistics are accessible from the `V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL` views.

> **See Also:**
>
> *Oracle Database Reference* for information about the `V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL` views

Active Session History Statistics

Any session that is connected to the database and is waiting for an event that does not belong to the Idle wait class is considered an active session. Oracle Database samples active sessions every second and stores the sampled data in a circular buffer in the shared global area (SGA).

The sampled session activity is accessible using the `V$ACTIVE_SESSION_HISTORY` view. Each session sample contains a set of rows and the `V$ACTIVE_SESSION_HISTORY` view returns one row for each active session per sample, starting with the latest session sample rows. Because the active session samples are stored in a circular buffer in the SGA, the greater the system activity, the smaller the number of seconds of session activity that can be stored. This means that the duration for which a session sample is displayed in the `V$` view is completely dependent on the level of database activity. Because the content of the `V$` view can become quite large during heavy system activity, only a portion of the session samples is written to disk.

By capturing only active sessions, a manageable set of data can be captured with its size being directly related to the work being performed, rather than the number of sessions allowed on the system. Active Session History (ASH) enables you to examine and perform detailed analysis on both current data in the `V$ACTIVE_SESSION_HISTORY` view and historical data in the `DBA_HIST_ACTIVE_SESS_HISTORY` view, often avoiding the need to replay the workload to trace additional performance information. ASH also contains execution plan information for each captured SQL statement. You can use this information to identify which part of SQL execution contributed most to the SQL elapsed time. The data present in ASH can be rolled up in various dimensions that it captures, including:

- SQL identifier of SQL statement
- SQL plan identifier and hash value of the SQL plan used to execute the SQL statement
- SQL execution plan information
- Object number, file number, and block number
- Wait event identifier and parameters
- Session identifier and session serial number
- Module and action name
- Client identifier of the session
- Service hash identifier
- Consumer group identifier

You can gather this information over a specified duration into an ASH report.

Active session history sampling is also available for Active Data Guard physical standby instances and Oracle Automatic Storage Management (Oracle ASM) instances. On these instances, the current session activity is collected and displayed in the `V$ACTIVE_SESSION_HISTORY` view, but not written to disk.

See Also:

- "Analyzing Sampled Data " for information about ASH reports
- *Oracle Data Guard Concepts and Administration* for information about Active Data Guard physical standby databases
- *Oracle Automatic Storage Management Administrator's Guide* for information about Oracle ASM instances

Wait Events Statistics

Wait events are statistics that are incremented by a server process or thread to indicate that it had to wait for an event to complete before processing could continue. Wait event data reveals various symptoms of problems that might be impacting performance, such as latch contention, buffer contention, and I/O contention.

To enable easier high-level analysis of wait events, Oracle Database groups events into the following classes:

- Administrative
- Application
- Cluster
- Commit
- Concurrency
- Configuration
- Idle
- Network

- Other
- Scheduler
- System I/O
- User I/O

The wait classes are based on a common solution that usually applies to fixing a problem with the particular wait event. For example, exclusive TX locks are generally an application-level issue and HW locks are generally a configuration issue. The following list includes common examples of wait events in some of the wait classes:

- Application: locks waits caused by row level locking or explicit lock commands
- Commit: waits for redo log write confirmation after a commit
- Idle: wait events that signify the session is inactive, such as `SQL*Net message from client`
- Network: waits for data to be sent over the network
- User I/O: wait for blocks to be read off a disk

Wait event statistics for a database instance include statistics for both background and foreground processes. Because tuning is typically focused in foreground activities, overall database instance activity is categorized into foreground and background statistics in the relevant `V$` views to facilitate tuning.

The `V$SYSTEM_EVENT` view shows wait event statistics for the foreground activities of a database instance and the wait event statistics for the database instance. The `V$SYSTEM_WAIT_CLASS` view shows these foreground and wait event statistics at the instance level after aggregating to wait classes. `V$SESSION_EVENT` and `V$SESSION_WAIT_CLASS` show wait event and wait class statistics at the session level.

> **See Also:**
>
> *Oracle Database Reference* for information about wait events

Session and System Statistics

A large number of cumulative database statistics on a system and session level are accessible using the `V$SYSSTAT` and `V$SESSTAT` views.

> **See Also:**
>
> *Oracle Database Reference* for information about the `V$SYSSTAT` and `V$SESSTAT` views

Interpreting Database Statistics

When initially examining performance data, you can formulate potential interpretations of the data by examining the database statistics. To ensure that your interpretation is

accurate, cross-check with other data to establish if a statistic or event is truly relevant. Because foreground activities are tunable, it is recommended to first analyze the statistics from foreground activities before analyzing the statistics from background activities.

The following sections provide tips for interpreting the various types of database statistics to measure database performance:

- Using Hit Ratios
- Using Wait Events with Timed Statistics
- Using Wait Events without Timed Statistics
- Using Idle Wait Events
- Comparing Database Statistics with Other Factors
- Using Computed Statistics

Using Hit Ratios

When tuning, it is common to compute a ratio that helps determine if a problem exists. Such ratios may include the buffer cache hit ratio, the soft-parse ratio, and the latch hit ratio. Do not use these ratios as definitive identifiers of whether a performance bottleneck exists. Instead, use them as indicators. To identify whether a performance bottleneck exists, examine other related performance data. For information about how to calculate the buffer cache hit ratio, see "Calculating the Buffer Cache Hit Ratio".

Using Wait Events with Timed Statistics

Setting `TIMED_STATISTICS` to `TRUE` at the instance level directs the database to gather wait time for events, in addition to available wait counts. This data is useful for comparing the total wait time for an event to the total elapsed time between the data collections. For example, if the wait event accounts for only 30 seconds out of a 2-hour period, then very little performance improvement can be gained by investigating this event, even if it is the highest ranked wait event when ordered by time waited. However, if the event accounts for 30 minutes of a 45-minute period, then the event is worth investigating. For information about wait events, see "Wait Events Statistics".

> **Note:**
>
> Timed statistics are automatically collected for the database if the initialization parameter `STATISTICS_LEVEL` is set to `TYPICAL` or `ALL`. If `STATISTICS_LEVEL` is set to `BASIC`, then you must set `TIMED_STATISTICS` to `TRUE` to enable collection of timed statistics. Note that setting `STATISTICS_LEVEL` to `BASIC` disables many automatic features and is not recommended.
>
> If you explicitly set `DB_CACHE_ADVICE`, `TIMED_STATISTICS`, or `TIMED_OS_STATISTICS`, either in the initialization parameter file or by using `ALTER_SYSTEM` or `ALTER SESSION`, then the explicitly set value overrides the value derived from `STATISTICS_LEVEL`.

> **See Also:**
>
> *Oracle Database Reference* for information about the `STATISTICS_LEVEL` initialization parameter

Using Wait Events without Timed Statistics

If `TIMED_STATISTICS` is set to `FALSE`, then the amount of time spent waiting for an event is not available. Therefore, it is only possible to order wait events by the number of times each event was waited for. Although the events with the largest number of waits might indicate a potential bottleneck, they might not be the main bottleneck. This situation can happen when an event is waited for a large number of times, but the total time waited for that event is small. Conversely, an event with fewer waits might be a bigger bottleneck if the wait time accounts for a significant proportion of the total wait time. Without the wait times to use for comparison, it is difficult to determine whether a wait event is worth investigating.

Using Idle Wait Events

Oracle Database uses some wait events to indicate whether the Oracle server process is idle. Typically, these events are of no value when investigating performance problems, and should be ignored when examining wait events.

Comparing Database Statistics with Other Factors

When evaluating statistics, it is important to consider other factors that may influence whether the statistic is of value. Such factors may include the user load and hardware capability. Even an event that had a wait of 30 minutes in a 45-minute period might not be indicative of a performance problem if you discover that there were 2000 users on the system, and the host hardware was a 64-node computer.

Using Computed Statistics

When interpreting computed statistics (such as rates, statistics normalized over transactions, or ratios), verify the computed statistic with the actual statistic counts. This comparison can confirm whether the derived rates are really of interest because small statistic counts usually can discount an unusual ratio. For example, on initial examination, a soft-parse ratio of 50% generally indicates a potential area for tuning. If, however, there was only one hard parse and one soft parse during the data collection interval, then the soft-parse ratio would be 50%, even though the statistic counts show this is not impacting performance. In this case, the ratio is not important due to the low raw statistic counts.

6

Gathering Database Statistics

This chapter describes how to gather database statistics for Oracle Database and contains the following topics:

About Gathering Database Statistics

Oracle Database automatically persists the cumulative and delta values for most of the statistics at all levels (except the session level) in the Automatic Workload Repository (AWR). This process is repeated on a regular time period and the results are captured in an AWR snapshot. The delta values captured by the snapshot represent the changes for each statistic over the time period.

A statistical baseline is a collection of statistic rates usually taken over a time period when the system is performing well at an optimal level. Use statistical baselines to diagnose performance problems by comparing statistics captured in a baseline to those captured during a period of poor performance. This enables you to identify specific statistics that may have increased significantly and could be the cause of the problem. AWR supports the capture of baseline data by enabling you to specify and preserve a pair or range of AWR snapshots as a baseline.

A metric is typically the rate of change in a cumulative statistic. You can measure this rate against a variety of units, including time, transactions, or database calls. For example, the number database calls per second is a metric. Metric values are exposed in some `V$` views, where the values are the averages over a fairly small time interval, typically 60 seconds. A history of recent metric values is available through `V$` views, and some data is also persisted by AWR snapshots.

The following sections describe various Oracle Database features that enable you to more effectively gather database statistics:

- Automatic Workload Repository
- Snapshots
- Baselines
- Space Consumption
- Adaptive Thresholds

Note:

- Data visibility and privilege requirements may differ when using AWR features with pluggable databases (PDBs). For information about how manageability features, including the AWR features, work in a multitenant container database (CDB), see *Oracle Multitenant Administrator's Guide*.
- License for Oracle Diagnostic Pack is required to use the AWR features described in this chapter.

Automatic Workload Repository

AWR collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. This gathered data is stored both in memory and in the database, and is displayed in both reports and views.

The statistics collected and processed by AWR include:

- Object statistics that determine both access and usage statistics of database segments
- Time model statistics based on time usage for activities, displayed in the `V$SYS_TIME_MODEL` and `V$SESS_TIME_MODEL` views
- Some of the system and session statistics collected in the `V$SYSSTAT` and `V$SESSTAT` views
- SQL statements that are producing the highest load on the system, based on criteria such as elapsed time and CPU time
- Active Session History (ASH) statistics, representing the history of recent sessions activity

See Also:

- "About Database Statistics" for information about the various types of database statistics
- *Oracle Database Reference* for more information about the views `V$SYS_TIME_MODEL`, `V$SESS_TIME_MODEL`, `V$SYSSTAT`, and `V$SESSTAT`

Snapshots

Snapshots are sets of historical data for specific time periods that are used for performance comparisons by Automatic Database Diagnostic Monitor (ADDM). By default, Oracle Database automatically generates snapshots of the performance data once every hour and retains the statistics in AWR for 8 days. You can also manually create snapshots or change the snapshot retention period, but it is usually not necessary.

AWR compares the difference between snapshots to determine which SQL statements to capture based on the effect on the system load. This reduces the number of SQL statements that must be captured over time. After the snapshots are created, ADDM analyzes the data captured in the snapshots to perform its performance analysis.

> **See Also:**
>
> "Managing Snapshots" for information about managing snapshots

Baselines

A baseline is a set of snapshots from a specific time period that is preserved for comparison with other snapshots when a performance problem occurs. The snapshots contained in a baseline are excluded from the automatic AWR purging process and are retained indefinitely.

There are several types of available baselines:

- Fixed Baselines
- Moving Window Baselines
- Baseline Templates

Fixed Baselines

A fixed baseline corresponds to a fixed, contiguous time period in the past that you specify. Before creating a fixed baseline, carefully consider the time period you choose as a baseline, because the baseline should represent the system operating at an optimal level. In the future, you can compare the baseline with other baselines or snapshots captured during periods of poor performance to analyze performance degradation over time.

> **See Also:**
>
> "Managing Baselines" for information about managing fixed baselines

Moving Window Baselines

A moving window baseline corresponds to all AWR data that exists within the AWR retention period. This is useful when using adaptive thresholds because the database can use AWR data in the entire AWR retention period to compute metric threshold values.

Oracle Database automatically maintains a system-defined moving window baseline. The default window size for the system-defined moving window baseline is the current AWR retention period, which by default is 8 days. If you are planning to use adaptive thresholds, then consider using a larger moving window—such as 30 days—to accurately compute threshold values. You can resize the moving window baseline by changing the number of days in the moving window to a value that is equal to or less than the number of days in the AWR retention period. Therefore, to increase the size of a moving window, you must first increase the AWR retention period accordingly.

> **See Also:**
>
> "Resizing the Default Moving Window Baseline" for information about resizing a moving window baseline

Baseline Templates

Baseline templates enable you to create baselines for a contiguous time period in the future. There are two types of baseline templates:

- Single Baseline Templates
- Repeating Baseline Templates

> **See Also:**
>
> "Managing Baseline Templates" for information about managing baseline templates

Single Baseline Templates

Use a single baseline template to create a baseline for a single contiguous time period in the future. This is useful if you know beforehand of a time period that you intend to capture in the future. For example, you may want to capture AWR data during a system test that is scheduled for the upcoming weekend. In this case, you can create a single baseline template to automatically capture the time period when the test occurs.

Repeating Baseline Templates

Use a repeating baseline template to create and drop baselines based on a repeating time schedule. This is useful if you want Oracle Database to automatically capture a contiguous time period on an ongoing basis. For example, you may want to capture AWR data during every Monday morning for a month. In this case, you can create a repeating baseline template to automatically create baselines on a repeating schedule for every Monday, and automatically remove older baselines after a specified expiration interval, such as one month.

Space Consumption

The space consumed by AWR is determined by several factors:

- Number of active sessions in the database at any given time
- Snapshot interval

  The snapshot interval determines the frequency at which snapshots are captured. A smaller snapshot interval increases the frequency, which increases the volume of data collected by AWR.
- Historical data retention period

The retention period determines how long this data is retained before being purged. A longer retention period increases the space consumed by AWR.

By default, Oracle Database captures snapshots once every hour and retains them in the database for 8 days. With these default settings, a typical system with an average of 10 concurrent active sessions can require approximately 200 to 300 MB of space for its AWR data.

To reduce AWR space consumption, increase the snapshot interval and reduce the retention period. When reducing the retention period, note that several Oracle Database self-managing features depend on AWR data for proper functioning. Not having enough data can affect the validity and accuracy of these components and features, including:

- Automatic Database Diagnostic Monitor (ADDM)
- SQL Tuning Advisor
- Undo Advisor
- Segment Advisor

If possible, Oracle recommends that you set the AWR retention period large enough to capture at least one complete workload cycle. If your system experiences weekly workload cycles, such as OLTP workload during weekdays and batch jobs during the weekend, then you do not need to change the default AWR retention period of 8 days. However, if your system is subjected to a monthly peak load during month-end book closing, then you may need to set the retention period to one month.

Under exceptional circumstances, you can disable automatic snapshot collection by setting the snapshot interval to 0. Under this condition, the automatic collection of the workload and statistical data is stopped, and most of the Oracle Database self-management functionality is not operational. In addition, you cannot manually create snapshots. For this reason, Oracle strongly recommends against disabling automatic snapshot collection.

Note:

Oracle Database uses the `SYSAUX` tablespace to store AWR data by default. Starting with Oracle Database 19c, you can specify any other tablespace to store the AWR data, so as to avoid overloading the `SYSAUX` tablespace.

See Also:

"Modifying Snapshot Settings" for information about changing the default values for the snapshot interval and retention period

Adaptive Thresholds

Adaptive thresholds enable you to monitor and detect performance issues, while minimizing administrative overhead. Adaptive thresholds automatically set warning and critical alert thresholds for some system metrics using statistics derived from metric values captured in the moving window baseline. The statistics for these thresholds are recomputed weekly and might result in new thresholds as system performance evolves over time. Additionally,

adaptive thresholds can compute different thresholds values for different times of the day or week based on periodic workload patterns.

For example, many databases support an online transaction processing (OLTP) workload during the day and batch processing at night. The performance metric for response time per transaction can be useful for detecting degradation in OLTP performance during the day. However, a useful OLTP threshold value is usually too low for batch workloads, where long-running transactions might be common. As a result, threshold values appropriate to OLTP might trigger frequent false performance alerts during batch processing. Adaptive thresholds can detect such a workload pattern and automatically set different threshold values for daytime and nighttime.

There are two types of adaptive thresholds:

- Percentage of Maximum Thresholds
- Significance Level Thresholds

Percentage of Maximum Thresholds

The threshold value for percentage of maximum thresholds is computed as a percentage multiple of the maximum value observed for the data in the moving window baseline.

Percentage of maximum thresholds are most useful when a system is sized for peak workloads, and you want to be alerted when the current workload volume approaches or exceeds previous high values. Metrics that have an unknown but definite limiting value are prime candidates for these settings. For example, the redo generated per second metric is typically a good candidate for a percentage of maximum threshold.

Significance Level Thresholds

The threshold value for significance level thresholds is set to a statistical percentile that represents how unusual it is to observe values above the threshold value based the data in the moving window baseline.

Significance level thresholds are most useful for metrics that exhibit statistically stable behavior when the system is operating normally, but might vary over a wide range when the system is performing poorly. For example, the response time per transaction metric should be stable for a well-tuned OLTP system, but may fluctuate widely when performance issues arise. Significance level thresholds are meant to generate alerts when conditions produce both unusual metric values and unusual system performance.

Significance level thresholds can be set to one of the following levels:

- High (.95)

  Only 5 in 100 observations are expected to exceed this value.
- Very High (.99)

  Only 1 in 100 observations are expected to exceed this value.
- Severe (.999)

  Only 1 in 1,000 observations are expected to exceed this value.
- Extreme (.9999)

  Only 1 in 10,000 observations are expected to exceed this value.

When you specify a significance level threshold, Oracle Database performs an internal calculation to set the threshold value. In some cases, Oracle Database cannot establish the threshold value at higher significance levels using the data in the baseline, and the significance level threshold is not set.

If you specified a Severe (.999) or Extreme (.9999) significance level threshold and are not receiving alerts as expected, try setting the significance level threshold to a lower value, such as Very High (.99) or High (.95). Alternatively, consider using a percentage of maximum threshold instead. If you change the threshold and find that you are receiving too many alerts, try increasing the number of occurrences to trigger an alert.

Note:

The primary interface for managing baseline metrics is Oracle Enterprise Manager Cloud Control (Cloud Control). To create an adaptive threshold for a baseline metric, use Cloud Control as described in *Oracle Database 2 Day + Performance Tuning Guide*.

See Also:

- "Moving Window Baselines" for information about moving window baselines
- "Managing Baselines" for information about managing baseline metrics

Managing the Automatic Workload Repository

This section describes how to manage AWR features of Oracle Database and contains the following topics:

- Enabling the Automatic Workload Repository
- Managing Snapshots
- Managing Baselines
- Managing Baseline Templates
- Transporting Automatic Workload Repository Data to Another System
- Using Automatic Workload Repository Views
- Managing Automatic Workload Repository in a Multitenant Environment
- Managing Automatic Workload Repository in Active Data Guard Standby Databases

See Also:

"Automatic Workload Repository" for a description of AWR

Enabling the Automatic Workload Repository

Gathering database statistics using AWR is enabled by default and is controlled by the `STATISTICS_LEVEL` initialization parameter.

To enable statistics gathering by AWR:

- Set the `STATISTICS_LEVEL` parameter to `TYPICAL` or `ALL`.

  The default setting for this parameter is `TYPICAL`.

Setting `STATISTICS_LEVEL` to `BASIC` disables many Oracle Database features, including AWR, and is not recommended. If `STATISTICS_LEVEL` is set to `BASIC`, you can still manually capture AWR statistics using the `DBMS_WORKLOAD_REPOSITORY` package. However, because in-memory collection of many system statistics—such as segments statistics and memory advisor information—will be disabled, the statistics captured in these snapshots may not be complete.

> **See Also:**
>
> *Oracle Database Reference* for information about the `STATISTICS_LEVEL` initialization parameter

Managing Snapshots

By default, Oracle Database generates snapshots once every hour, and retains the statistics in the workload repository for 8 days. When necessary, you can manually create or drop snapshots and modify snapshot settings.

This section describes how to manage snapshots and contains the following topics:

- User Interfaces for Managing Snapshots
- Creating Snapshots
- Dropping Snapshots
- Modifying Snapshot Settings

> **See Also:**
>
> "Snapshots" for information about snapshots

User Interfaces for Managing Snapshots

The primary interface for managing snapshots is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, you should manage snapshots using Cloud Control.

If Cloud Control is unavailable, then manage snapshots using the `DBMS_WORKLOAD_REPOSITORY` package in the command-line interface. The DBA role is required to invoke the `DBMS_WORKLOAD_REPOSITORY` procedures.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Creating Snapshots

By default, Oracle Database automatically generates snapshots once every hour. However, you may want to manually create snapshots to capture statistics at times different from those of the automatically generated snapshots.

Creating Snapshots Using the Command-Line Interface

To manually create snapshots, use the `CREATE_SNAPSHOT` procedure. The following example shows a `CREATE_SNAPSHOT` procedure call.

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
END;
/
```

In this example, a snapshot is created immediately on the local database instance. To view information about an existing snapshot, use the `DBA_HIST_SNAPSHOT` view.

> **Note:**
>
> You can specify value for the `flush_level` parameter of the `CREATE_SNAPSHOT` procedure to either TYPICAL or ALL. The default value for the flush level is TYPICAL.
>
> The flush level signifies the breadth and depth of the AWR statistics to be captured. If you want to capture all the AWR statistics, then set the flush level to ALL. If you want to skip few AWR statistics, such as, SQL statistics, segment statistics, and files and tablespace statistics for performance reasons, then set the flush level to TYPICAL.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package
> - *Oracle Database Reference* for information about the `DBA_HIST_SNAPSHOT` view

Dropping Snapshots

By default, Oracle Database automatically purges snapshots that have been stored in AWR for over 8 days. However, you may want to manually drop a range of snapshots to free up space.

Dropping Snapshots Using the Command-Line Interface

To manually drop a range of snapshots, use the `DROP_SNAPSHOT_RANGE` procedure. The following example shows a `DROP_SNAPSHOT_RANGE` procedure call.

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE (low_snap_id  => 22,
                                                high_snap_id => 32,
                                                dbid         => 3310949047);
END;
/
```

In the example, snapshots with snapshot IDs ranging from 22 to 32 are dropped immediately from the database instance with the database identifier of `3310949047`. Any ASH data that were captured during this snapshot range are also purged.

> **Tip:**
>
> To determine which snapshots to drop, use the `DBA_HIST_SNAPSHOT` view to review the existing snapshots

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package
> - *Oracle Database Reference* for information about the `DBA_HIST_SNAPSHOT` view

Modifying Snapshot Settings

You can adjust the interval, retention period, and number of top SQL to flush for snapshot generation, but note that this can affect the precision of the Oracle Database diagnostic tools.

Modifying Snapshot Settings Using the Command-Line Interface

You can modify snapshot settings using the following parameters of the `DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS` procedure:

Parameter	Description
`INTERVAL`	This setting affects how often the database automatically generates snapshots.

Parameter	Description
RETENTION	This setting affects how long the database stores snapshots in AWR.
TOPNSQL	This setting affects the number of top SQL to flush for each SQL criteria (elapsed time, CPU time, parse calls, sharable memory, and version count). This setting is not affected by the statistics/ flush level and overrides the system default behavior for AWR SQL collection. It is possible to set the value for this setting to MAXIMUM to capture the complete set of SQL in the shared SQL area, though doing so (or by setting the value to a very high number) may lead to possible space and performance issues because there will be more data to collect and store.

The following example shows how to modify snapshot settings using the DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS procedure:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS( retention  => 43200,
                                                     interval   => 30,
                                                     topnsql    => 100,
                                                     dbid       => 3310949047);
END;
/
```

In this example, snapshot settings for the database with the database identifier of 3310949047 are modified as follows:

- The retention period is specified as 43200 minutes (30 days).
- The interval between each snapshot is specified as 30 minutes.
- The number of top SQL to flush for each SQL criteria is specified as 100.

To get information about the current snapshot settings for your database, use the DBA_HIST_WR_CONTROL view as shown in the following example:

```
SQL> select snap_interval, retention from DBA_HIST_WR_CONTROL;

SNAP_INTERVAL               RETENTION
-------------------------------------
+00000 01:00:00.0        +00008 00:00:00.0
```

The snap_interval and retention values are displayed in the format:

```
+[days] [hours]:[minutes]:[seconds].[nanoseconds]
```

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS` procedure
> - *Oracle Database Reference* for more information about the `DBA_HIST_WR_CONTROL` view

Managing Baselines

By default, Oracle Database automatically maintains a system-defined moving window baseline. When necessary, you can manually create, drop, or rename a baseline and view the baseline threshold settings. Additionally, you can manually resize the window size of the moving window baseline.

This section describes how to manage baselines and contains the following topics:

- User Interface for Managing Baselines
- Creating a Baseline
- Dropping a Baseline
- Renaming a Baseline
- Displaying Baseline Metrics
- Resizing the Default Moving Window Baseline

> **See Also:**
>
> "Baselines" for information about baselines

User Interface for Managing Baselines

The primary interface for managing baselines is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, manage baselines using Cloud Control.

If Cloud Control is unavailable, then manage baselines using the `DBMS_WORKLOAD_REPOSITORY` package in the command-line interface. The DBA role is required to invoke the `DBMS_WORKLOAD_REPOSITORY` procedures.

> **See Also:**
>
> - *Oracle Database 2 Day + Performance Tuning Guide* for more information about managing baselines using Cloud Control
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Creating a Baseline

By default, Oracle Database automatically maintains a system-defined moving window baseline. However, you may want to manually create a fixed baseline that represents the system operating at an optimal level, so you can compare it with other baselines or snapshots captured during periods of poor performance.

To create baselines using command-line interface, use the `CREATE_BASELINE` procedure as shown in the following example:

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (start_snap_id => 270,
                                             end_snap_id   => 280,
                                             baseline_name => 'peak baseline',
                                             dbid          => 3310949047,
                                             expiration    => 30);
END;
/
```

In this example, a baseline is created on the database instance with the database identifier of `3310949047` with the following settings:

- The start snapshot sequence number is 270.
- The end snapshot sequence number is 280.
- The name of baseline is `peak baseline`.
- The expiration of the baseline is 30 days.

Oracle Database automatically assigns a unique ID to the new baseline when the baseline is created.

> **Tip:**
>
> To determine the range of snapshots to include in a baseline, use the `DBA_HIST_SNAPSHOT` view to review the existing snapshots

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package
> - *Oracle Database Reference* for information about the `DBA_HIST_SNAPSHOT` view

Dropping a Baseline

To conserve disk space, consider periodically dropping a baseline that is no longer being used. The snapshots associated with a baseline are retained indefinitely until you explicitly drop the baseline or the baseline has expired.

To drop a baseline using command-line interface, use the `DROP_BASELINE` procedure as shown in the following example:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE (baseline_name => 'peak baseline',
                                          cascade       => FALSE,
                                          dbid          => 3310949047);
END;
/
```

In the example, the baseline `peak baseline` is dropped from the database instance with the database identifier of `3310949047` and the associated snapshots are preserved.

Tip:

To determine which baseline to drop, use the `DBA_HIST_BASELINE` view to review the existing baselines.

Tip:

To drop the associated snapshots along with the baseline, set the `cascade` parameter to `TRUE`.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information about the `DBMS_WORKLOAD_REPOSITORY` package

Renaming a Baseline

To rename a baseline using command-line interface, use the `RENAME_BASELINE` procedure. The following example shows a `RENAME_BASELINE` procedure call.

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.RENAME_BASELINE (old_baseline_name => 'peak
baseline',
                                             new_baseline_name => 'peak
mondays',
                                             dbid              => 3310949047);
END;
/
```

In this example, the name of the baseline on the database instance with the database identifier of `3310949047` is renamed from `peak baseline` to `peak mondays`.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Displaying Baseline Metrics

When used with adaptive thresholds, a baseline contains AWR data that the database can use to compute metric threshold values.

To display the summary statistics for metric values in a baseline period using the command-line interface, use the `SELECT_BASELINE_METRICS` function:

```
DBMS_WORKLOAD_REPOSITORY.SELECT_BASELINE_METRICS (baseline_name IN VARCHAR2,
                                                  dbid          IN NUMBER DEFAULT NULL,
                                                  instance_num  IN NUMBER DEFAULT NULL)
   RETURN awr_baseline_metric_type_table PIPELINED;
```

> **See Also:**
>
> - "Adaptive Thresholds" for information about baseline metric thresholds
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Resizing the Default Moving Window Baseline

By default, Oracle Database automatically maintains a system-defined moving window baseline. The default window size for the system-defined moving window baseline is the current AWR retention period, which by default is 8 days. In certain circumstances, you may want to modify the window size of the default moving window baseline, such as increasing its size to more accurately compute threshold values for adaptive thresholds.

To modify the window size of the default moving window baseline using the command-line interface, use the `MODIFY_BASELINE_WINDOW_SIZE` procedure as shown in the following example:

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.MODIFY_BASELINE_WINDOW_SIZE (window_size => 30,
                                                          dbid        => 3310949047);
END;
/
```

In this example, the default moving window is resized to 30 days on the database instance with the database identifier of `3310949047`.

> **Note:**
>
> The window size must be set to a value that is equal to or less than the value of the AWR retention setting. To set a window size that is greater than the current AWR retention period, you must first increase the value of the `retention` parameter as described in "Modifying Snapshot Settings".

> **See Also:**
>
> - "Moving Window Baselines" for information about moving window baselines
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Managing Baseline Templates

Baseline templates enable you to automatically create baselines to capture specified time periods in the future. This section describes how to manage baseline templates and contains the following topics:

- User Interfaces for Managing Baseline Templates
- Creating a Single Baseline Template
- Creating a Repeating Baseline Template
- Dropping a Baseline Template

> **See Also:**
>
> "Baseline Templates" for information about baseline templates

User Interfaces for Managing Baseline Templates

The primary interface for managing baseline templates is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, manage baseline templates using Cloud Control.

If Cloud Control is unavailable, then manage baseline templates using the `DBMS_WORKLOAD_REPOSITORY` package in the command-line interface. The DBA role is required to invoke the `DBMS_WORKLOAD_REPOSITORY` procedures.

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for more information about managing baseline templates using Cloud Control

Creating a Single Baseline Template

You can use a single baseline template to create a baseline during a single, fixed time interval in the future. For example, you can create a single baseline template to generate a baseline that is captured on April 2, 2012 from 5:00 p.m. to 8:00 p.m.

To create a single baseline template using command-line interface, use the `CREATE_BASELINE_TEMPLATE` procedure as shown in the following example:

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (start_time    => '2012-04-02
17:00:00 PST',
                                                       end_time      => '2012-04-02
20:00:00 PST',
                                                       baseline_name =>
'baseline_120402',
                                                       template_name =>
'template_120402',
                                                       expiration    => 30,
                                                       dbid          => 3310949047);

END;
/
```

In this example, a baseline template named `template_120402` is created that will generate a baseline named `baseline_120402` for the time period from 5:00 p.m. to 8:00 p.m. on April 2, 2012 on the database with a database ID of `3310949047`. The baseline will expire after 30 days.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Creating a Repeating Baseline Template

You can use a repeating baseline template to automatically create baselines that repeat during a particular time interval over a specific period in the future. For example, you can create a repeating baseline template to generate a baseline that repeats every Monday from 5:00 p.m. to 8:00 p.m. for the year 2012.

To create a repeating baseline template using command-line, use the `CREATE_BASELINE_TEMPLATE` procedure as shown in the following example:

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (day_of_week => 'monday',
                                                       hour_in_day => 17,
                                                       duration => 3, expiration => 30,
```

```
                                                    start_time => '2012-04-02
17:00:00 PST',
                                                    end_time => '2012-12-31
20:00:00 PST',
                                                    baseline_name_prefix =>
'baseline_2012_mondays_',
                                                    template_name =>
'template_2012_mondays',
                                                    dbid => 3310949047);
END;
/
```

In this example, a baseline template named `template_2012_mondays` is created that will generate a baseline on every Monday from 5:00 p.m. to 8:00 p.m. beginning on April 2, 2012 at 5:00 p.m. and ending on December 31, 2012 at 8:00 p.m. on the database with a database ID of `3310949047`. Each of the baselines will be created with a baseline name with the prefix `baseline_2012_mondays_` and will expire after 30 days.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Dropping a Baseline Template

Periodically, you may want to remove baselines templates that are no longer used to conserve disk space.

To drop a baseline template using command-line, use the `DROP_BASELINE_TEMPLATE` procedure as shown in the following example:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE_TEMPLATE (template_name =>
'template_2012_mondays',
                                                   dbid          => 3310949047);
END;
/
```

In this example, the baseline template named `template_2012_mondays` is dropped from the database instance with the database identifier of `3310949047`.

> **Tip:**
>
> To determine which baseline template to drop, use the `DBA_HIST_BASELINE_TEMPLATE` view to review the existing baseline templates.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPOSITORY` package

Transporting Automatic Workload Repository Data to Another System

Oracle Database enables you to transport AWR data between systems. This is useful in cases where you want to use a separate system to perform analysis of AWR data, so as to reduce the overhead caused by performance analysis on a production system.

To transport AWR data from one system to another, first export the AWR data from the database on the source system, and then import it into the database on the target system.

This section contains the following topics:

- "Exporting AWR Data"
- "Importing AWR Data"

Exporting AWR Data

The `awrextr.sql` script exports AWR data for a range of snapshots from the database into a Data Pump export file. After it is created, you can transport this dump file to another database where you can import the exported AWR data. To run the `awrextr.sql` script, you must be connected to the database as the `SYS` user.

To export AWR data:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/awrextr.sql
   ```

   A list of the databases in the AWR schema is displayed.

2. Specify the database from which AWR data needs to be exported:

   ```
   Enter value for db_id: 1377863381
   ```

   In this example, the database with the database identifier of `1377863381` is specified.

3. Specify the number of days for which you want to view all the snapshot IDs:

   ```
   Enter value for num_days: 2
   ```

   In this example, all the snapshots captured in the last 2 days are displayed.

4. Define the range of snapshots for which AWR data needs to be exported by specifying the beginning and the ending snapshot IDs:

   ```
   Enter value for begin_snap: 30
   Enter value for end_snap: 40
   ```

   In this example, the snapshot ID of 30 is specified as the beginning snapshot, and the snapshot ID of 40 is specified as the ending snapshot.

   A list of directory objects is displayed.

5. Specify the directory object pointing to the directory where the export dump file needs to be stored:

```
Enter value for directory_name: DATA_PUMP_DIR
```

In this example, the directory object `DATA_PUMP_DIR` is specified that points to the directory `ORACLE_HOME/rdbms/log`, where `ORACLE_HOME` is `/u01/app/oracle/product/`*`database_release_number`*`/dbhome_1`.

6. Specify a name for the export dump file without the file extension. By default, the file extension of `.dmp` is used.

```
Enter value for file_name: awrdata_30_40
```

In this example, an export dump file named `awrdata_30_40.dmp` is created in the directory specified in the directory object `DATA_PUMP_DIR`:

```
Dump file set for SYS.SYS_EXPORT_TABLE_01 is:
/u01/app/oracle/product/database_release_number/dbhome_1/rdbms/log/
awrdata_30_40.dmp
Job "SYS"."SYS_EXPORT_TABLE_01" successfully completed at 08:58:20
```

Depending on the amount of AWR data that must be exported, the AWR export operation may take a while to complete. After the dump file is created, you can use Data Pump to transport the file to another system.

> **See Also:**
>
> *Oracle Database Utilities* for information about using Data Pump

Importing AWR Data

After the export dump file is transported to the target system, import the exported AWR data using the `awrload.sql` script. The `awrload.sql` script creates a staging schema where the snapshot data is transferred from the Data Pump file into the database. The data is then transferred from the staging schema into the appropriate AWR tables. To run the `awrload.sql` script, you must be connected to the database as the `SYS` user.

To import AWR data:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/awrload.sql
```

A list of directory objects is displayed.

2. Specify the directory object pointing to the directory where the export dump file is located:

```
Enter value for directory_name: DATA_PUMP_DIR
```

In this example, the directory object `DATA_PUMP_DIR` is specified that points to the directory where the export dump file is located.

3. Specify the name of the export dump file without the file extension. By default, the file extension of `.dmp` is used.

```
Enter value for file_name: awrdata_30_40
```

In this example, the export dump file named `awrdata_30_40.dmp` is selected.

4. Specify the name of the staging schema where the AWR data needs to be imported:

```
Enter value for schema_name: AWR_STAGE
```

In this example, a staging schema named `AWR_STAGE` is created.

5. Specify the default tablespace for the staging schema:

```
Enter value for default_tablespace: SYSAUX
```

In this example, the `SYSAUX` tablespace is specified.

6. Specify the temporary tablespace for the staging schema:

```
Enter value for temporary_tablespace: TEMP
```

In this example, the `TEMP` tablespace is specified.

7. First the AWR data is imported into the `AWR_STAGE` schema and then it is transferred to the AWR tables in the `SYS` schema:

```
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Completed 113 CONSTRAINT objects in 11 seconds
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
Completed 1 REF_CONSTRAINT objects in 1 seconds
Job "SYS"."SYS_IMPORT_FULL_03" successfully completed at 09:29:30
... Dropping AWR_STAGE user
End of AWR Load
```

Depending on the amount of AWR data that must be imported, the AWR import operation may take a while to complete. After AWR data is imported, the staging schema will be dropped automatically.

Using Automatic Workload Repository Views

Typically, you would view AWR data using Oracle Enterprise Manager Cloud Control (Cloud Control) or AWR reports. However, you can also view historical data stored in the AWR using the following `DBA_HIST` views.

> **Note:**
>
> In a multitenant environment, these `DBA_HIST` views can also be interchanged with the `AWR_ROOT` views and `AWR_PDB` views at the CDB level and the PDB level respectively. For example, you can use the `AWR_PDB_ACTIVE_SESS_HISTORY` view for retrieving the AWR data about the active session history at the PDB level, which is equivalent to the `DBA_HIST_ACTIVE_SESS_HISTORY` view in an independent database in a non-multitenant environment. The `AWR_PDB` views will not show any AWR data, if the PDB level snapshots have not been collected.

Table 6-1 DBA_HIST Views

DBA_HIST View	Description
DBA_HIST_ACTIVE_SESS_HISTORY	Displays the history of the contents of the in-memory active session history for recent system activity.
DBA_HIST_BASELINE	Displays information about the baselines captured on the system, such as the time range of each baseline and the baseline type.
DBA_HIST_BASELINE_DETAILS	Displays details about a specific baseline.
DBA_HIST_BASELINE_TEMPLATE	Displays information about the baseline templates used by the system to generate baselines.
DBA_HIST_CON_SYS_TIME_MODEL	Displays historical system time model statistics, including OLAP timed statistics.
DBA_HIST_CON_SYSMETRIC_HIST	Displays the historical information about the system metric values.
DBA_HIST_CON_SYSMETRIC_SUMM	Displays history of the statistical summary of all the metric values in the system metrics for the long duration (60 seconds) group.
DBA_HIST_CON_SYSSTAT	Displays historical system statistics, including OLAP kernel statistics.
DBA_HIST_CON_SYSTEM_EVENT	Displays historical information about the total waits for an event.
DBA_HIST_DATABASE_INSTANCE	Displays information about the database environment.
DBA_HIST_DB_CACHE_ADVICE	Displays historical predictions of the number of physical reads for the cache size corresponding to each row.
DBA_HIST_DISPATCHER	Displays historical information for each dispatcher process at the time of the snapshot.
DBA_HIST_DYN_REMASTER_STATS	Displays statistical information about the dynamic remastering process.
DBA_HIST_IOSTAT_DETAIL	Displays historical I/O statistics aggregated by file type and function.
DBA_HIST_RSRC_PDB_METRIC	Displays historical information about the Resource Manager metrics for pluggable databases (PDBs) for the past one hour.
DBA_HIST_RSRC_METRIC	Displays historical information about the Resource Manager metrics for consumer groups for the past one hour.
DBA_HIST_SHARED_SERVER_SUMMARY	Displays historical information for shared servers, such as shared server activity, common queues and dispatcher queues.
DBA_HIST_SNAPSHOT	Displays information on snapshots in the system.
DBA_HIST_SQL_PLAN	Displays the SQL execution plans.
DBA_HIST_WR_CONTROL	Displays the settings for controlling AWR.
DBA_HIST_WR_SETTINGS	Displays the settings and metadata of the AWR.
DBA_HIST_PROCESS_WAITTIME	Displays CPU and wait time for a process type.

See Also:

Oracle Database Reference for more information about the DBA_HIST views

Managing Automatic Workload Repository in a Multitenant Environment

A centralized Automatic Workload Repository (AWR) stores the performance data related to CDB and PDBs in a multitenant environment.

CDBs and individual PDBs can store, view, and manage AWR data. You can take an AWR snapshot at the CDB level or at the PDB level.

This section contains the following topics:

Categorization of AWR Data in a Multitenant Environment

In a multitenant environment, AWR data falls into different categories.

The categories are as follows:

- General AWR data

  This data has no security implications. It is safe to be shared among all tenants in a CDB. This data is accessible by all PDBs and is captured in both CDB-level and PDB-level snapshots. Examples of general AWR data include the names of statistics, latches, and parameters.

- AWR data for a CDB

  This category aggregates data for all tenants in a CDB. This data contains the status of the database as a whole and is useful only for the CDB administrator. This data is captured only in the CDB-level snapshots.

- AWR data for individual PDBs

  This data describes the individual PDBs in a CDB. It shows container-specific data that represents the contribution of each individual PDB to the whole database instance. Therefore, this data is useful for both the CDB and the PDB administrators. This data is captured in both CDB-level and PDB-level snapshots.

AWR Data Storage and Retrieval in a Multitenant Environment

This section describes the process of managing snapshots, and exporting and importing AWR data in a multitenant environment.

Managing Snapshots

Starting with Oracle Database 12*c* Release 2 (12.2), you can take an AWR snapshot at a CDB-level, that is, on a CDB root, as well as at a PDB-level, that is, on an individual PDB. By default, the CDB-level snapshot data is stored in the `SYSAUX` tablespace of a CDB root and the PDB-level snapshot data is stored in the `SYSAUX` tablespace of a PDB.

A CDB-level snapshot contains information about the CDB statistics as well as all the PDB statistics, such as ASH, SQL statistics, and file statistics. The CDB administrator can perform CDB-specific management operations, such as setting AWR data retention period, setting snapshot schedule, taking manual snapshots, and purging snapshot data for a CDB root.

A PDB-level snapshot contains the PDB statistics and also some global statistics that can be useful for diagnosing the performance problems related to the PDB. The PDB administrator can perform PDB-specific management operations, such as setting AWR data retention period, setting snapshot schedule, taking manual snapshots, and purging snapshot data for a PDB.

The CDB-level and PDB-level snapshot operations, such as creating snapshots and purging snapshots, can be performed in either the *automatic mode* or the *manual mode*.

The automatic snapshot operations are scheduled, so that they get executed automatically at a particular time. The `AWR_PDB_AUTOFLUSH_ENABLED` initialization parameter enables you to specify whether to enable or disable automatic snapshots for all the PDBs in a CDB or for individual PDBs in a CDB. The automatic snapshot operations are enabled by default for a CDB, but are disabled by default for a PDB. To enable automatic snapshots for a PDB, the PDB administrator must connect to that PDB, set the value for the `AWR_PDB_AUTOFLUSH_ENABLED` parameter to `true`, and set the snapshot generation interval to a value greater than 0.

See Also:

Oracle Database Reference for more information about the `AWR_PDB_AUTOFLUSH_ENABLED` initialization parameter

The manual snapshot operations are explicitly initiated by users. The automatic snapshots and manual snapshots capture the same AWR information. Oracle recommends to generally use manual snapshots for a PDB. You should enable automatic snapshots only selectively for a PDB for performance reasons.

The primary interface for managing snapshots is Oracle Enterprise Manager Cloud Control (Cloud Control). If Cloud Control is not available, then you can use the procedures in the `DBMS_WORKLOAD_REPOSITORY` package to manage snapshots. The Oracle DBA role is required to use the procedures in the `DBMS_WORKLOAD_REPOSITORY` package. The SQL procedures to create, drop, and modify snapshots for a CDB root and a PDB are the same as that for a non-CDB. These SQL procedures perform their operations on the local database by default, if the target database information is not provided in their procedure call.

Note:

- The PDB-level snapshots have unique snapshot IDs and are not related to the CDB-level snapshots.
- The plugging and unplugging operations of a PDB in a CDB do not affect the AWR data stored on a PDB.
- The CDB administrator can use the *PDB lockdown profiles* to disable the AWR functionality for a PDB by executing the following SQL statement on that PDB:

  ```
  SQL> alter lockdown profile profile_name disable
  feature=('AWR_ACCESS');
  ```

  Once the AWR functionality is disabled on a PDB, snapshot operations cannot be performed on that PDB.

  The AWR functionality can be enabled again for a PDB by executing the following SQL statement on that PDB:

  ```
  SQL> alter lockdown profile profile_name enable
  feature=('AWR_ACCESS');
  ```

- Snapshot data is stored in the `SYSAUX` tablespace of a CDB and a PDB by default. Starting with Oracle Database 19c, you can specify any other tablespace to store snapshot data for a CDB and a PDB by modifying snapshot settings.

See Also:

- "Creating Snapshots"
- "Dropping Snapshots"
- "Modifying Snapshot Settings"
- *Oracle Database Security Guide* for more information about the PDB lockdown profiles

Exporting and Importing AWR Data

The process of exporting and importing AWR data for a CDB root and a PDB in a multitenant environment is similar to the process of exporting and importing AWR data for a non-CDB.

> **See Also:**
>
> - "Exporting AWR Data" for information about exporting AWR data from an Oracle database
> - "Importing AWR Data" for information about importing AWR data into an Oracle database

Viewing AWR Data in a Multitenant Environment

You can view the AWR data in a multitenant environment using various Oracle Database reports and views.

AWR Reports

The primary interface for generating AWR reports is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, generate AWR reports using Cloud Control.

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for more information about generating AWR report using Cloud Control

If Cloud Control is unavailable, then you can generate the AWR reports by running SQL scripts as described below. The DBA role is required to run these scripts.

- You can generate a CDB-specific AWR report from a CDB root that shows the *global system data* statistics for the whole multitenant environment. You can generate this AWR report using the SQL scripts described in the section "Generating an AWR Report for the Local Database".
- You can generate a PDB-specific AWR report from a PDB that shows the statistics related to that PDB. You can generate this AWR report using the SQL scripts described in the section "Generating an AWR Report for the Local Database".
- You can generate a PDB-specific AWR report from a CDB root that shows the statistics related to a specific PDB. You can generate this AWR report using the SQL scripts described in the section "Generating an AWR Report for a Specific Database".

AWR Views

The following table lists the Oracle Database views for accessing the AWR data stored on the CDB root and the individual PDBs in a multitenant environment.

> **See Also:**
>
> "Using Automatic Workload Repository Views" for more information about these AWR views

Table 6-2 Views for Accessing AWR Data in a Multitenant Environment

Views	Description
`DBA_HIST` Views	• The `DBA_HIST` views show the AWR data present only on the CDB root. • When the `DBA_HIST` views are accessed from a CDB root, they show all the AWR data stored on the CDB root. • When the `DBA_HIST` views are accessed from a PDB, they show the subset of the CDB root AWR data, which is specific to that PDB.
`DBA_HIST_CON` Views	• The `DBA_HIST_CON` views are similar to the `DBA_HIST` views, but they provide more fine grained information about each container, and thus, they have more data than the `DBA_HIST` views. • The `DBA_HIST_CON` views show the AWR data present only on the CDB root. • When the `DBA_HIST_CON` views are accessed from a CDB root, they show all the AWR data stored on the CDB root. • When the `DBA_HIST_CON` views are accessed from a PDB, they show the subset of the CDB root AWR data, which is specific to that PDB.
`AWR_ROOT` Views	• The `AWR_ROOT` views are available starting with Oracle Database 12*c* Release 2 (12.2) and are available only in the Multitenant environment. • The `AWR_ROOT` views are equivalent to the `DBA_HIST` views. • The `AWR_ROOT` views show the AWR data present only on the CDB root. • When the `AWR_ROOT` views are accessed from a CDB root, they show all the AWR data stored on the CDB root. • When the `AWR_ROOT` views are accessed from a PDB, they show the subset of the CDB root AWR data, which is specific to that PDB.
`AWR_PDB` Views	• The `AWR_PDB` views are available starting with Oracle Database 12*c* Release 2 (12.2). • The `AWR_PDB` views show the local AWR data present on a CDB root or a PDB. • When the `AWR_PDB` views are accessed from a CDB root, they show the AWR data stored on the CDB root. • When the `AWR_PDB` views are accessed from a PDB, they show the AWR data stored on that PDB.
`CDB_HIST` Views	• The `CDB_HIST` views show the AWR data stored on the PDBs. • When the `CDB_HIST` views are accessed from a CDB root, they show the union of the AWR data stored on all the PDBs. • When the `CDB_HIST` views are accessed from a PDB, they show the AWR data stored on that PDB.

Managing Automatic Workload Repository in Active Data Guard Standby Databases

Starting with Oracle Database 12*c* Release 2 (12.2), Automatic Workload Repository (AWR) data can be captured for Active Data Guard (ADG) standby databases. This feature enables analyzing any performance-related issues for ADG standby databases.

AWR snapshots for ADG standby databases are called *remote snapshots*. A database node, called *destination*, is responsible for storing snapshots that are collected from remote ADG standby database nodes, called *sources*.

A destination can be either an ADG primary database or a non-ADG database. If a destination is an ADG primary database, then it is also a source database, and its snapshots are *local snapshots*.

A source is identified by a unique name or *source name* by which it is known to a destination.

You can assign a name to a destination node or a source node during its configuration. Otherwise, the value of the initialization parameter `DB_UNIQUE_NAME` is assigned as a name for a node.

Each source must have two database links, a destination-to-source database link and a source-to-destination database link. These database links are configured for each source during the ADG deployment. You must manually reconfigure these database links after certain ADG events, such as failovers, switchovers, and addition and removal of hosts, so that the database applications continue functioning properly after these events.

You can take the remote snapshots either automatically at scheduled time intervals or manually. The remote snapshots are always started by the destination node. After the destination initiates the snapshot creation process, sources *push* their snapshot data to the destination using database links. The snapshot data or *AWR data* stored on the destination can be accessed using AWR reports, Oracle Database import and export functions, and user-defined queries. The Automatic Database Diagnostic Monitor (ADDM) application can use the AWR data for analyzing any database performance-related issues.

Destination Database Responsibilities

A destination database manages the following tasks:

- Registering sources
- Assigning unique identifier for each source
- Creating database links between destination and sources
- Scheduling and initiating automatic snapshots for sources
- Managing destination workload by coordinating snapshots among sources
- Managing snapshot settings for each source
- Assigning identifiers to newly generated snapshots
- Partitioning the AWR tables

- Storing the performance data in the local AWR
- Purging the AWR data of destination and sources

source Database Responsibilities

A source database manages the following tasks:

- Storing its performance data in the local AWR
- Sending its AWR data to the destination
- Responding to service requests from the destination
- Extracting the AWR data from the destination

Major Steps for Managing AWR in ADG Standby Databases

The following are the major steps for managing AWR in ADG standby databases:

1. Configuring the Remote Management Framework (RMF)
2. Managing Snapshots for Active Data Guard Standby Databases
3. Viewing AWR Data in Active Data Guard Standby Databases

> **Note:**
>
> Before you start configuring AWR for ADG environment, make sure that the database links for all the ADG standby databases are already configured during the ADG deployment.

Configuring the Remote Management Framework (RMF)

The Remote Management Framework (RMF) is an architecture for capturing performance statistics (AWR data) in an Oracle database.

> **Note:**
>
> RMF can be used only for ADG standby databases and standalone databases.

The RMF *topology* is a centralized architecture that consists of all the participating database nodes along with their metadata and connection information. The RMF topology has one database node, called *destination*, which is responsible for storing and managing performance data (AWR data) that is collected from the database nodes, called *sources*. A *candidate destination* is a source that can be configured in such way that it can replace the original destination, when the original destination is unavailable or is downgraded. A topology can have only one destination, and one or more candidate destinations.

Each database node in a topology must be assigned a unique name. This can be done using the procedure `DBMS_UMF.configure_node()` during configuring a node. If the name for a node is not provided in this procedure, then the value of the initialization parameter `DB_UNIQUE_NAME` is used as the name for a node.

The database nodes in a topology communicate with each other using database links. The database links between destination to source and source to destination must be created for each ADG standby database during the ADG deployment.

A *service* is an application running on a topology. For example, an AWR service running on a topology enables remote AWR snapshots for all the database nodes in that topology.

The RMF APIs are the PL/SQL procedures and functions that can be used to configure the RMF topology. The RMF APIs are declared in the PL/SQL package `DBMS_UMF`.

Note:

- The `SYS$UMF` user is the default database user that has all the privileges to access the system-level RMF views and tables. All the AWR related operations in RMF can be performed only by the `SYS$UMF` user. The `SYS$UMF` user is locked by default and it must be unlocked before deploying the RMF topology.
- You need to provide password for the `SYS$UMF` user when creating database links in the RMF topology. If the password for the `SYS$UMF` user is changed, all the database links in the RMF topology must be recreated.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about the `DBMS_UMF` package

Setting Up the RMF Topology

You need to set up the RMF topology for collecting performance statistics for an Oracle database.

The following are the prerequisites for setting up the RMF topology:

- You must create destination to source and source to destination database links for all the database nodes to be registered in the RMF topology. This setup should be done during the ADG deployment.

The following are the steps for setting up the RMF topology:

1. Configure database nodes to add to the topology.
2. Create the topology.
3. Register database nodes with the topology.
4. (Optional) Register database links between the nodes in the topology. This configuration is required when a destination becomes unavailable and a candidate destination needs to connect to the remaining nodes in the topology using database links.

Example for Setting Up the RMF Topology

In this example, the three database nodes `T`, `S0`, and `S1` are added to the topology `Topology_1`. Node `T` is the destination node and nodes `S0` and `S1` are the source nodes. Node `S1` is a candidate destination, that is, when the original destination `T` is not available, node `S1` becomes the new destination. The AWR service is enabled for all the sources in the topology.

Assume that the following database links are already created during the ADG deployment:

- `DBLINK_T_to_S0`: Database link from `T` to `S0`.
- `DBLINK_T_to_S1`: Database link from `T` to `S1`.
- `DBLINK_S0_to_T`: Database link from `S0` to `T`.
- `DBLINK_S0_to_S1`: Database link from `S0` to `S1`.
- `DBLINK_S1_to_T`: Database link from `S1` to `T`.
- `DBLINK_S1_to_S0`: Database link from `S1` to `S0`.

The following is a sample code for setting up the RMF topology:

```
/* Configure the nodes T, S0, and S1 by executing these procedures */

/* Execute this procedure on node T */
SQL> exec DBMS_UMF.configure_node ('T');

/* Execute this procedure on node S0 */
SQL> exec DBMS_UMF.configure_node ('S0', 'DBLINK_S0_to_T');

/* Execute this procedure on node S1 */
SQL> exec DBMS_UMF.configure_node ('S1', 'DBLINK_S1_to_T');

/* Execute all the following procedures on the destination node T */

/* Create the topology 'Topology_1' */
SQL> exec DBMS_UMF.create_topology ('Topology_1');

/* Register the node S0 with the topology 'Topology_1' */
SQL> exec DBMS_UMF.register_node ('Topology_1',
                                  'S0',
                                  'DBLINK_T_to_S0',
                                  'DBLINK_S0_to_T',
                                  'TRUE'  /* Set it as a source */,
                                  'FALSE' /* Set it as not a candidate destination */);

/* Register the node S1 with the topology 'Topology_1' */
SQL> exec DBMS_UMF.register_node ('Topology_1',
                                  'S1',
                                  'DBLINK_T_to_S1',
                                  'DBLINK_S1_to_T',
                                  'TRUE'  /* Set it as a source */,
                                  'TRUE'  /* Set it as a candidate destination */);

/* Register the database links between the nodes S0 and S1 in the topology 'Topology_1'.
 * When destination T is unavailable at the time of failover, the source S0 can connect
 * to the candidate destination S1 using this database link.
```

```
 */
SQL> exec DBMS_UMF.create_link ('Topology_1',
                                'S0',
                                'S1',
                                'DBLINK_S0_to_S1',
                                'DBLINK_S1_to_S0');

/* Enable the AWR service on the node S0 in the topology 'Topology_1' */
SQL> exec DBMS_WORKLOAD_REPOSITORY.register_remote_database(node_name=>'S0');

/* Enable the AWR service on the node S1 in the topology 'Topology_1' */
SQL> exec DBMS_WORKLOAD_REPOSITORY.register_remote_database(node_name=>'S1');
```

> **Note:**
>
> The AWR service can be disabled for a node using the procedure:
>
> ```
> SQL> exec
> DBMS_WORKLOAD_REPOSITORY.unregister_remote_database(node_name)
> ```

Managing ADG Role Transition

An ADG role transition occurs when the ADG Primary or original destination fails (*failover* event) or when an ADG standby database or candidate destination takes over the role of the ADG Primary during the maintenance phase (*switchover* event).

Oracle recommends that you perform the following configuration steps before making the role change, that is, before making the candidate destination as the new destination due to the failover or switchover event:

1. Create database links between the sources and the candidate destination. This configuration must be done for all the sources by executing the following procedure on each source:

   ```
   SQL> EXEC DBMS_UMF.CREATE_LINK (topology name,
                                   source name,
                                   candidate destination name,
                                   source to candidate destination database link,
                                   candidate destination to source database link);
   ```

   > **Note:**
   >
   > Oracle recommends that you create database links among all the nodes in a topology to avoid any unanticipated issues that may arise at the time of role change.

2. Take an AWR snapshot on the candidate destination.

> **Note:**
>
> To generate an AWR report for the candidate destination after the role change, take at least one snapshot for the candidate destination before the role change.

3. Restart the candidate destination as well as all the sources.

After completing the preceding configuration steps, you can make the role change by executing the following procedure on the candidate destination:

```
SQL> EXEC DBMS_UMF.SWITCH_DESTINATION(topology name, force_switch=>FALSE);
```

> **Note:**
>
> Oracle recommends that you do not take any snapshots for the sources during the role transition period. After the role change process is complete by executing the `DBMS_UMF.SWITCH_DESTINATION` procedure, you can take snapshots for the sources. If you want to generate AWR reports for the sources after the role change, then you must choose only those snapshots that were taken after the role change.

Getting the Details of Registered RMF Topologies

The RMF views described below show the configuration information about all the registered RMF topologies in a multi-database environment.

Table 6-3 RMF Views

RMF View	Description
`DBA_UMF_TOPOLOGY`	Shows all the registered topologies in a multi-database environment. Each topology has a topology name, a destination ID, and topology state. To enable RMF, the topology state of at least one topology should be `ACTIVE`.
`DBA_UMF_REGISTRATION`	Shows all the registered nodes in all the topologies in a multi-database environment.
`DBA_UMF_LINK`	Shows all the registered database links in all the topologies in a multi-database environment.
`DBA_UMF_SERVICE`	Shows all the registered services in all the topologies in a multi-database environment.

> **See Also:**
>
> *Oracle Database Reference* for more information about these RMF views

Managing Snapshots for Active Data Guard Standby Databases

The AWR snapshots for ADG standby databases are called *remote* snapshots. Similar to local AWR snapshots, remote AWR snapshots can be generated automatically at scheduled time intervals or can be generated manually. The *Push-on-Demand* mechanism is used for generating remote snapshots, where the snapshots generation process is initiated by the destination, which then instructs the sources to start *pushing* the snapshot data to the destination over database links. The destination periodically initiates automatic snapshots based on the snapshot time interval configured for each of the sources.

> **Note:**
>
> The destination is responsible for purging the expired remote snapshots based on the snapshot data or *AWR data* retention settings for individual sources. Purging of locally generated snapshots occurs as part of the regularly scheduled purging process. By default, Oracle Database automatically purges snapshots that have been stored in AWR for over 8 days. The partitioning of AWR table for remote snapshots is done in the same way as that of the local snapshots.

Creating, Modifying, and Deleting Remote Snapshots

The APIs for creating, modifying, and deleting *remote* snapshots are same as that for the *local* snapshots.

> **Note:**
>
> For creating *remote* snapshots, you can also use the `DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT` API. This API works similar to the *local* snapshot creation API `DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT`, but it takes the additional parameter of RMF topology name.

> **See Also:**
>
> - "Creating Snapshots"
> - "Modifying Snapshot Settings"
> - "Dropping Snapshots"
> - *Oracle Database PL/SQL Packages and Types Reference* for the syntax of the `DBMS_WORKLOAD_REPOSITORY.CREATE_REMOTE_SNAPSHOT` API.

Managing Baselines for Remote Snapshots

The APIs for managing baselines for *remote* snapshots are same as that for the *local* snapshots.

> **See Also:**
>
> "Managing Baselines"

Exporting and Importing Remote Snapshots

> **Note:**
>
> You cannot execute the AWR export and import scripts related to remote snapshots on an ADG standby database, that is, on a source database. Always execute these scripts on a destination database.

The process of exporting and importing AWR data for *remote* snapshots is same as that for the *local* snapshots. Starting with Oracle Database 12*c* Release 2 (12.2), the AWR data export and import scripts `awrextr.sql` and `awrload.sql` use the *source name* identifier to distinguish snapshots originating from a particular source. A source name is stored in a dump file during an export operation and is used as a default source name during an import operation.

> **See Also:**
>
> "Transporting Automatic Workload Repository Data to Another System" for information about exporting and importing AWR data for local snapshots.

Exporting Remote Snapshots Using the awrextr.sql Script

The process of exporting *remote* snapshots is similar to exporting *local* snapshots using the `awrextr.sql` script described in the section "Exporting AWR Data" with the following differences:

- The default export log file directory is the same as that of the dump file, but you can also specify any other directory for an export log file.
- The `.dmp` suffix can be specified to the name of the dump file to export.
- The export script displays the values of `SOURCE_DBID` and `SOURCE_NAME` columns of AWR tables before prompting for the *Mapped Database ID* value to export.

Importing Remote Snapshots Using the awrload.sql Script

The process of importing *remote* snapshots is similar to importing *local* snapshots using the `awrload.sql` script described in the section "Importing AWR Data" with the following differences:

- The default import log file directory is the same as that of the dump file, but you can also specify any other directory for an import log file. This is particularly useful when the dump file resides in a read-only directory.
- The `.dmp` suffix can be specified to the name of the dump file to import.
- The import script uses the values of `SOURCE_DBID` and `SOURCE_NAME` columns present in the dump file to determine the appropriate *Mapped Database ID* to use for storing the snapshot data in AWR.

> **Note:**
>
> The snapshot import operation is not affected by the version of the Oracle database from which the snapshot dump was generated.

Viewing AWR Data in Active Data Guard Standby Databases

You can view the AWR data stored in the ADG standby databases using Oracle supplied AWR views and AWR reports.

Viewing AWR Data Using AWR Views

You can view the historical data stored in AWR using the *`DBA_HIST`* views described in the section "Using Automatic Workload Repository Views".

> **Note:**
>
> Starting with Oracle Database 12*c* Release 2 (12.2), the view `DBA_HIST_DATABASE_INSTANCE` contains the column `DB_UNIQUE_NAME` to support AWR for ADG standby databases. The column `DB_UNIQUE_NAME` stores the unique identifier of a source by which it is known to the destination.

Viewing AWR Data Using AWR Reports

You can view the performance statistics related to ADG standby databases using AWR reports. The primary interface for generating AWR reports is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, generate AWR reports using Cloud Control. If Cloud Control is unavailable, then generate AWR reports using the Oracle supplied SQL scripts. The DBA role is required to run these scripts.

The AWR data can be queried for a particular source using the source name-mapped database ID pair. The mapped database ID is similar to the database identifier (DBID) that is used by AWR to identify a database instance and is stored in the `DBID` column in the AWR tables. The AWR DBID value is derived as follows for the ADG standby databases:

- For a destination, the AWR DBID value is the value of `V$DATABASE.CON_DBID`.
- For a source, the AWR DBID value is the value of `DBMS_UMF.GET_NODE_ID_LOCAL()` or the value of the column `NODE_ID` in the `DBA_UMF_REGISTRATION` view.

As snapshot IDs are not unique across sources, the pair of snapshot ID-mapped database ID identifies a snapshot for a particular source.

> **See Also:**
>
> "Generating an AWR Report for a Specific Database" for information about generating AWR reports using Oracle supplied SQL scripts.

Generating Automatic Workload Repository Reports

An AWR report shows data captured between two snapshots (or two points in time). AWR reports are divided into multiple sections. The content of the report contains the workload profile of the system for the selected range of snapshots. The HTML report includes links that can be used to navigate quickly between sections.

> **Note:**
>
> If you run a report on a database that does not have any workload activity during the specified range of snapshots, then calculated percentages for some report statistics can be less than 0 or greater than 100. This result means that there is no meaningful value for the statistic.

This section describes how to generate AWR reports and contains the following topics:

- User Interface for Generating an AWR Report
- Generating an AWR Report Using the Command-Line Interface

User Interface for Generating an AWR Report

The primary interface for generating AWR reports is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, generate AWR reports using Cloud Control.

If Cloud Control is unavailable, then generate AWR reports by running SQL scripts. The DBA role is required to run these scripts.

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for more information about generating AWR reports using Cloud Control

Generating an AWR Report Using the Command-Line Interface

This section describes how to generate AWR reports by running SQL scripts in the command-line interface. The DBA role is required to run these scripts. Click on an appropriate task link in the following table for the detailed steps to generate the required AWR report.

Table 6-4 SQL Scripts for Generating AWR Reports

Task	SQL Script	Description
Generating an AWR Report for the Local Database	`awrrpt.sql`	Generates an AWR report in HTML or text format that displays statistics from a range of snapshot IDs in the local database instance.
Generating an AWR Report for a Specific Database	`awrrpti.sql`	Generates an AWR report in HTML or text format that displays statistics from a range of snapshot IDs in a specific database instance.
Generating an AWR Report for the Local Database in Oracle RAC	`awrgrpt.sql`	Generates an AWR report in HTML or text format that displays statistics from a range of snapshot IDs in the local database instance in an Oracle RAC environment.
Generating an AWR Report for a Specific Database in Oracle RAC	`awrgrpti.sql`	Generates an AWR report in HTML or text format that displays statistics from a range of snapshot IDs in a specific database instance in an Oracle RAC environment.
Generating an AWR Report for a SQL Statement on the Local Database	`awrsqrpt.sql`	Generates an AWR report in HTML or text format that displays statistics for a particular SQL statement from a range of snapshot IDs in the local database instance.
Generating an AWR Report for a SQL Statement on a Specific Database	`awrsqrpi.sql`	Generates an AWR report in HTML or text format that displays statistics for a particular SQL statement from a range of snapshot IDs in a specific database instance.

Generating an AWR Report for the Local Database

The `awrrpt.sql` SQL script generates an HTML or text report that displays statistics from a range of snapshot IDs.

To generate an AWR report on the local database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/awrrpt.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: text
   ```

   In this example, a text report is chosen.

3. Specify the number of days for which you want to list snapshot IDs.

   ```
   Enter value for num_days: 2
   ```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

4. Specify a beginning and ending snapshot ID for the workload repository report:

   ```
   Enter value for begin_snap: 150
   Enter value for end_snap: 160
   ```

In this example, the snapshot with a snapshot ID of 150 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 160 is selected as the ending snapshot.

5. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrrpt_1_150_160
```

In this example, the default name is accepted and an AWR report named `awrrpt_1_150_160` is generated.

Generating an AWR Report for a Specific Database

The `awrrpti.sql` SQL script generates an HTML or text report that displays statistics from a range of snapshot IDs using a specific database instance. This script enables you to specify a database identifier and instance for which the AWR report will be generated.

To generate an AWR report on a specific database instance using the command-line interface:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/awrrpti.sql
```

2. Specify whether you want an HTML or a text report:

```
Enter value for report_type: text
```

In this example, a text report is chosen.

A list of available database identifiers and instance numbers are displayed:

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
 3309173529        1 MAIN         main         examp1690
 3309173529        1 TINT251      tint251      samp251
```

3. Enter the values for the database identifier (`dbid`) and instance number (`inst_num`):

```
Enter value for dbid: 3309173529
Using 3309173529 for database Id
Enter value for inst_num: 1
```

> **Note:**
>
> For an ADG standby database, the value for `dbid` can be determined as follows:
>
> - For a Destination node, use the value of `v$database.con_dbid` .
> - For a Source node, use the value of `dbms_umf.get_node_id_local()`.

4. Specify the number of days for which you want to list snapshot IDs.

```
Enter value for num_days: 2
```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

5. Specify a beginning and ending snapshot ID for the workload repository report:

```
Enter value for begin_snap: 150
Enter value for end_snap: 160
```

In this example, the snapshot with a snapshot ID of 150 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 160 is selected as the ending snapshot.

6. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrrpt_1_150_160
```

In this example, the default name is accepted and an AWR report named `awrrpt_1_150_160` is generated on the database instance with a database ID value of `3309173529`.

Generating an AWR Report for the Local Database in Oracle RAC

The `awrgrpt.sql` SQL script generates an HTML or text report that displays statistics from a range of snapshot IDs using the current database instance in an Oracle Real Application Clusters (Oracle RAC) environment.

> **Note:**
>
> In an Oracle RAC environment, Oracle recommends generating an HTML report (instead of a text report) because it is much easier to read.

To generate an AWR report for Oracle RAC on the local database instance using the command-line interface:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/awrgrpt.sql
```

2. Specify whether you want an HTML or a text report:

```
Enter value for report_type: html
```

In this example, an HTML report is chosen.

3. Specify the number of days for which you want to list snapshot IDs.

```
Enter value for num_days: 2
```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last day are displayed.

4. Specify a beginning and ending snapshot ID for the workload repository report:

```
Enter value for begin_snap: 150
Enter value for end_snap: 160
```

In this example, the snapshot with a snapshot ID of 150 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 160 is selected as the ending snapshot.

5. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrrpt_rac_150_160.html
```

In this example, the default name is accepted and an AWR report named `awrrpt_rac_150_160.html` is generated.

Generating an AWR Report for a Specific Database in Oracle RAC

The `awrgrpti.sql` SQL script generates an HTML or text report that displays statistics from a range of snapshot IDs using specific databases instances running in an Oracle RAC environment. This script enables you to specify database identifiers and a comma-delimited list of database instances for which the AWR report will be generated.

> **Note:**
>
> In an Oracle RAC environment, Oracle recommends generating an HTML report (instead of a text report) because it is much easier to read.

To generate an AWR report for Oracle RAC on a specific database instance using the command-line interface:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/awrgrpti.sql
```

2. Specify whether you want an HTML or a text report:

```
Enter value for report_type: html
```

In this example, an HTML report is chosen.

A list of available database identifiers and instance numbers are displayed:

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
 3309173529        1 MAIN         main         examp1690
 3309173529        1 TINT251      tint251      samp251
 3309173529        2 TINT251      tint252      samp252
```

3. Enter the value for the database identifier (`dbid`):

```
Enter value for dbid: 3309173529
Using 3309173529 for database Id
```

4. Enter the value for the instance numbers (`instance_numbers_or_all`) of the Oracle RAC instances you want to include in the report:

```
Enter value for instance_numbers_or_all: 1,2
```

5. Specify the number of days for which you want to list snapshot IDs.

```
Enter value for num_days: 2
```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

6. Specify a beginning and ending snapshot ID for the workload repository report:

```
Enter value for begin_snap: 150
Enter value for end_snap: 160
```

In this example, the snapshot with a snapshot ID of 150 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 160 is selected as the ending snapshot.

7. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrrpt_rac_150_160.html
```

In this example, the default name is accepted and an AWR report named `awrrpt_rac_150_160.html` is generated on the database instance with a database ID value of `3309173529`.

Generating an AWR Report for a SQL Statement on the Local Database

The `awrsqrpt.sql` SQL script generates an HTML or text report that displays statistics of a particular SQL statement from a range of snapshot IDs. Run this report to inspect or debug the performance of a SQL statement.

To generate an AWR report for a SQL statement on the local database instance using the command-line interface:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/awrsqrpt.sql
```

2. Specify whether you want an HTML or a text report:

```
Enter value for report_type: html
```

In this example, an HTML report is chosen.

3. Specify the number of days for which you want to list snapshot IDs.

```
Enter value for num_days: 1
```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

4. Specify a beginning and ending snapshot ID for the workload repository report:

```
Enter value for begin_snap: 146
Enter value for end_snap: 147
```

In this example, the snapshot with a snapshot ID of 146 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 147 is selected as the ending snapshot.

5. Specify the SQL ID of a particular SQL statement to display statistics:

```
Enter value for sql_id: 2b064ybzkwf1y
```

In this example, the SQL statement with a SQL ID of `2b064ybzkwf1y` is selected.

6. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrrpt_1_146_147.html
```

In this example, the default name is accepted and an AWR report named `awrrpt_1_146_147` is generated.

Generating an AWR Report for a SQL Statement on a Specific Database

The `awrsqrpi.sql` SQL script generates an HTML or text report that displays statistics of a particular SQL statement from a range of snapshot IDs using a specific database instance.This script enables you to specify a database identifier and instance for which the AWR report will be generated. Run this report to inspect or debug the performance of a SQL statement on a specific database and instance.

To generate an AWR report for a SQL statement on a specific database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/awrsqrpi.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: html
   ```

   In this example, an HTML report is chosen.

   A list of available database identifiers and instance numbers are displayed:

   ```
   Instances in this Workload Repository schema
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      DB Id    Inst Num DB Name      Instance     Host
   ----------- -------- ------------ ------------ ------------
    3309173529        1 MAIN         main         examp1690
    3309173529        1 TINT251      tint251      samp251
   ```

3. Enter the values for the database identifier (`dbid`) and instance number (`inst_num`):

   ```
   Enter value for dbid: 3309173529
   Using 3309173529 for database Id
   Enter value for inst_num: 1
   Using 1 for instance number
   ```

4. Specify the number of days for which you want to list snapshot IDs.

   ```
   Enter value for num_days: 1
   ```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

5. Specify a beginning and ending snapshot ID for the workload repository report:

   ```
   Enter value for begin_snap: 146
   Enter value for end_snap: 147
   ```

   In this example, the snapshot with a snapshot ID of 146 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 147 is selected as the ending snapshot.

6. Specify the SQL ID of a particular SQL statement to display statistics:

   ```
   Enter value for sql_id: 2b064ybzkwf1y
   ```

In this example, the SQL statement with a SQL ID of `2b064ybzkwf1y` is selected.

7. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrrpt_1_146_147.html
```

In this example, the default name is accepted and an AWR report named `awrrpt_1_146_147` is generated on the database instance with a database ID value of `3309173529`.

Generating Performance Hub Active Report

Performance Hub feature of EM Express provides an active report with a consolidated view of all performance data for a specified time period. The report is fully interactive; its contents are saved in a HTML file, which you can access offline using a web browser.

> **See Also:**
>
> *Oracle Database 2 Day DBA* for more information about Performance Hub feature of EM Express

This section describes how to generate Performance Hub active report and contains the following topics:

- Overview of Performance Hub Active Report
- Command-Line User Interface for Generating a Performance Hub Active Report
- Generating a Performance Hub Active Report Using a SQL Script

Overview of Performance Hub Active Report

Performance Hub active report enables you to view all performance data available for a time period that you specify. Different tabs are available in the Performance Hub, depending on whether real-time or historical data is selected for the time period. When real-time data is selected, more granular data is presented, because real-time data for the last hour is displayed. When historical data is selected, more detailed data is presented, but the data points are averaged out to the Automatic Workload Repository (AWR) interval for the selected time period.

This section describes Performance Hub active report and contains the following topics:

- About Performance Hub Active Report Tabs
- About Performance Hub Active Report Types

About Performance Hub Active Report Tabs

Performance Hub active report contains interactive tabs that enable you to view and navigate through performance data categorized into various performance areas.

The tabs contained in a Performance Hub active report include the following:

- Summary

  The Summary tab provides an overview of system performance, including resource consumption, average active sessions, and load profile information. This tab is available for real-time data as well as historical data.

- Activity

  The Activity tab displays ASH analytics. This tab is available for real-time data as well as historical data.

- Workload

  The Workload tab displays metric information about the workload profile, such as call rates, logon rate, and top SQL. This tab is available for real-time data as well as historical data.

- RAC

  The RAC tab displays metrics specific to Oracle RAC, such as the number of global cache blocks received and the average block latency. This tab is only available in Oracle RAC environments. This tab is available for real-time data as well as historical data.

- Monitored SQL

  The Monitored SQL tab displays information about monitored SQL statements. This tab is available for real-time data as well as historical data.

- ADDM

  The ADDM tab displays information for ADDM analysis tasks and Real-Time ADDM analysis reports. This tab is available for real-time data as well as historical data.

- Current ADDM Findings

  The Current ADDM Findings tab displays a real-time analysis of system performance for the past 5 minutes. This tab is only available if the specified time period for the Performance Hub active report is within the past hour. This tab is available only for real-time data.

- Database time

  The Database Time tab displays wait events by category for various metrics. This tab is available only for historical data.

- Resources

  The Resources tab displays operating system and I/O usage statistics. This tab is available only for historical data.

- System Statistics

  The System Statistics tab displays database and system statistics. This tab is available only for historical data.

About Performance Hub Active Report Types

You can choose the level of details displayed within each tab of the Performance Hub active report by selecting the report type.

The available report types for the Performance Hub active report include the following:

- Basic

  Only the basic information for all the tabs is saved to the report.

- Typical

  In addition to the information saved in the basic report type, the SQL Monitor information for the top SQL statements contained in the Monitored SQL tab and the ADDM reports are saved to the report.

- All

  In addition to the information saved in the typical report type, the SQL Monitor information for all SQL statements contained in the Monitored SQL tab and all detailed reports for all tabs are saved to the report.

Command-Line User Interface for Generating a Performance Hub Active Report

You can generate a Performance Hub active report using the command-line interface in one of two ways:

- Using a SQL script, as described in "Generating a Performance Hub Active Report Using a SQL Script".
- Using the `DBMS_PERF` package, as described in *Oracle Database PL/SQL Packages and Types Reference*.

Generating a Performance Hub Active Report Using a SQL Script

This section describes how to generate Performance Hub active report by running the `perfhubrpt.sql` SQL script in the command-line interface. The DBA role is required to run this script.

To generate a Performance Hub active report:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/perfhubrpt.sql
   ```

2. Specify the desired report type:

   ```
   Please enter report type: typical
   ```

   For information about the available report types, see "About Performance Hub Active Report Types".

3. Enter the value for the database identifier of the database you want to use:

   ```
   Please enter database ID: 3309173529
   ```

   To use the local database, enter a null value (the default value). If you specify a database identifier for a database other than the local database, then the Performance Hub active report is generated from imported AWR data.

4. Enter the value for the instance number of the database instance you want to use:

   ```
   Please enter instance number: all instances
   ```

   To specify all instances, enter `all instances` (the default value).

5. Enter the desired time period by specifying an end time and a start time in the format of `dd:mm:yyyy hh:mi:ss`:

```
Please enter end time in format of dd:mm:yyyy hh24:mi:ss: 03:04:2014 17:00:00
Please enter start time in format of dd:mm:yyyy hh24:mi:ss: 03:04:2014 16:00:00
```

6. Enter a report name, or accept the default report name:

```
Enter value for report_name: my_perfhub_report.html
```

In this example, a Performance Hub active report named `my_perfhub_report` is generated on all database instances with a database ID value of `3309173529` for the specified time period from 4:00 p.m. to 5:00 p.m on April 3, 2014.

7

Automatic Performance Diagnostics

This chapter describes Oracle Database automatic features for performance diagnosing and tuning.

This chapter contains the following topics:

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for information about using Oracle Enterprise Manager Cloud Control (Cloud Control) to diagnose and tune the database with the Automatic Database Diagnostic Monitor

Overview of the Automatic Database Diagnostic Monitor

The Automatic Workload Repository (AWR) stores performance related statics for an Oracle database. The Automatic Database Diagnostic Monitor (ADDM) is a diagnostic tool that analyzes the AWR data on a regular basis, locates root causes of any performance problems, provides recommendations for correcting the problems, and identifies non-problem areas of the system. Because AWR is a repository of historical performance data, ADDM can analyze performance issues after the event, often saving time and resources in reproducing a problem.

In most cases, ADDM output should be the first place that a DBA looks when notified of a performance problem. ADDM provides the following benefits:

- Automatic performance diagnostic report every hour by default
- Problem diagnosis based on decades of tuning expertise
- Time-based quantification of problem impacts and recommendation benefits
- Identification of root cause, not symptoms
- Recommendations for treating the root causes of problems
- Identification of non-problem areas of the system
- Minimal overhead to the system during the diagnostic process

Tuning is an iterative process, and fixing one problem can cause the bottleneck to shift to another part of the system. Even with the benefit of ADDM analysis, it can take multiple tuning cycles to reach acceptable system performance. ADDM benefits apply beyond

production systems; on development and test systems, ADDM can provide an early warning of performance issues.

This section contains the following topics:

> **Note:**
>
> Data visibility and privilege requirements may differ when using ADDM features with pluggable databases (PDBs). For information about how manageability features, including ADDM features, work in a multitenant container database (CDB), see *Oracle Multitenant Administrator's Guide*.

ADDM Analysis

An ADDM analysis can be performed on a pair of AWR snapshots and a set of instances from the same database. The pair of AWR snapshots define the time period for analysis, and the set of instances define the target for analysis.

If you are using Oracle Real Application Clusters (Oracle RAC), then ADDM has three analysis modes:

- Database

  In Database mode, ADDM analyzes all instances of the database.

- Instance

  In Instance mode, ADDM analyzes a particular instance of the database.

- Partial

  In Partial mode, ADDM analyzes a subset of all database instances.

If you are not using Oracle RAC, then ADDM can only function in Instance mode because only one instance of the database exists.

An ADDM analysis is performed each time an AWR snapshot is taken and the results are saved in the database. The time period analyzed by ADDM is defined by the last two snapshots (the last hour by default). ADDM will always analyze the specified instance in Instance mode. For non-Oracle RAC or single instance environments, the analysis performed in the Instance mode is the same as a database-wide analysis. If you are using Oracle RAC, then ADDM also analyzes the entire database in Database mode, as described in "Using ADDM with Oracle Real Application Clusters".

After ADDM completes its analysis, you can view the ADDM results using either Cloud Control, or `DBMS_ADDM` package subprograms, or `DBA_ADDM_*` and `DBA_ADVISOR_*` views.

ADDM analysis is performed top down, first identifying symptoms, and then refining them to reach the root causes of performance problems. The goal of the analysis is to reduce a single throughput metric called `DB time`. `DB time` is the fundamental measure of database performance, and is the cumulative time spent by the database in processing user requests. It includes wait time and CPU time of all non-idle user foreground sessions. `DB time` is displayed in the `V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL` views.

By reducing `DB time`, the database is able to support more user requests using the same resources, which increases throughput. The problems reported by ADDM are sorted by the amount of `DB time` they are responsible for. System areas that are not responsible for a significant portion of `DB time` are reported as non-problem areas.

The types of problems that ADDM considers include the following:

- CPU bottlenecks - Is the system CPU bound by Oracle Database or some other application?
- Undersized Memory Structures - Are the Oracle Database memory structures, such as the SGA, PGA, and buffer cache, adequately sized?
- I/O capacity issues - Is the I/O subsystem performing as expected?
- High-load SQL statements - Are there any SQL statements which are consuming excessive system resources?
- High-load PL/SQL execution and compilation, and high-load Java usage
- Oracle RAC specific issues - What are the global cache hot blocks and objects; are there any interconnect latency issues?
- Sub-optimal use of Oracle Database by the application - Are there problems with poor connection management, excessive parsing, or application level lock contention?
- Database configuration issues - Is there evidence of incorrect sizing of log files, archiving issues, excessive checkpoints, or sub-optimal parameter settings?
- Concurrency issues - Are there buffer busy problems?
- Hot objects and top SQL for various problem areas

> **Note:**
>
> This is not a comprehensive list of all problem types that ADDM considers in its analysis.

ADDM also documents the non-problem areas of the system. For example, wait event classes that are not significantly impacting the performance of the system are identified and removed from the tuning consideration at an early stage, saving time and effort that would be spent on items that do not impact overall system performance.

> **See Also:**
>
> - *Oracle Database Reference* for information about the `V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL` views
> - "Time Model Statistics" for a discussion of time model statistics and `DB time`
> - *Oracle Database Concepts* for information about server processes

Using ADDM with Oracle Real Application Clusters

If you are using Oracle RAC, then run ADDM in Database analysis mode to analyze the throughput performance of all instances of the database. In Database mode, ADDM considers DB time as the sum of the database time for all database instances. Using the Database analysis mode enables you to view all findings that are significant to the entire database in a single report, instead of reviewing a separate report for each instance.

The Database mode report includes findings about database resources (such as I/O and interconnect). The report also aggregates findings from the various instances if they are significant to the entire database. For example, if the CPU load on a single instance is high enough to affect the entire database, then the finding appears in the Database mode analysis, which points to the particular instance responsible for the problem.

> **See Also:**
>
> *Oracle Real Application Clusters Administration and Deployment Guide* for information about using ADDM with Oracle RAC

Using ADDM in a Multitenant Environment

Starting with Oracle Database 12c, ADDM is enabled by default in the root container of a multitenant container database (CDB). Starting with Oracle Database 19c, you can also use ADDM in a pluggable database (PDB).

In a CDB, ADDM works in the same way as it works in a non-CDB, that is, the ADDM analysis is performed each time an AWR snapshot is taken on a CDB root or a PDB, and the ADDM results are stored on the same database system where the snapshot is taken. The time period analyzed by ADDM is defined by the last two snapshots (the last hour by default).

After ADDM completes its analysis, you can view the ADDM results using any of the following methods:

- Using Enterprise Manager Cloud Control (Cloud Control)
- Using the `DBA_ADDM_*` and `DBA_ADVISOR_*` views

Note:

- ADDM is enabled by default in a CDB root.
- ADDM does not work in a PDB by default, because automatic AWR snapshots are disabled by default in a PDB. To use ADDM in a PDB, you must enable automatic AWR snapshots in the PDB.
- A user whose current container is the CDB root can view ADDM results for the entire CDB. The ADDM results can include information about multiple PDBs. ADDM results related to a PDB are not included if the PDB is unplugged. The ADDM results stored on the CDB root cannot be viewed when the current container is a PDB.
- ADDM results on a PDB provide only PDB-specific findings and recommendations. A user whose current container is a PDB can view ADDM results for the current PDB only. The ADDM results exclude findings that apply to the CDB as a whole, for example, I/O problems relating to the buffer cache size.
- Enabling AWR snapshots on a PDB does not change the ADDM report on the CDB root.
- AWR data on a PDB cannot be accessed from the CDB root.

PDB-Level ADDM Restrictions

Unlike in a non-CDB, ADDM does not report the following issues in a PDB, because these issues apply to a CDB as a whole and do not apply to an individual PDB:

- I/O problems due to:
  - undersized buffer cache
  - undersized streams pool
  - excessive temporary writes
  - excessive checkpoint writes
  - excessive undo writes
  - excessive PQ checkpoint writes
  - excessive truncate writes
  - excessive tablespace DDL checkpoint
  - I/O capacity limit
- SQL hard parsing issues due to:
  - cursor aging
  - out-of-memory failed parse
- SGA sizing issues

ADDM also does not report the following issues in a PDB, because these issues cannot be resolved at a PDB level:

- Cluster messaging related issues, such as network latency, congestion, contention, and lost blocks

- Log file switch waits on archiving and on checkpoint incomplete
- Too many free-buffer waits
- Contention on log buffer waits
- Waits due to CPU bottleneck
- Operating system VM paging
- Session slot wait event
- CPU quantum wait event
- RMAN related wait events, such as PQ queued wait event, PGA limit wait event, and I/O queue wait event

See Also:

- "Enabling ADDM in a Pluggable Database" for information about how to enable ADDM in a PDB
- "ADDM Views" for more information about the ADDM views `DBA_ADDM_*` and `DBA_ADVISOR_*`
- "Diagnosing Database Performance Problems with ADDM" for information about how to run ADDM in an Oracle database using the `DBMS_ADDM` package subprograms
- *Oracle Database 2 Day + Performance Tuning Guide* for information about how to run ADDM in an Oracle database using Cloud Control

Enabling ADDM in a Pluggable Database

ADDM does not work in a pluggable database (PDB) by default, because automatic AWR snapshots are disabled by default in a PDB. To use ADDM in a PDB, you must enable automatic AWR snapshots in the PDB by setting the `AWR_PDB_AUTOFLUSH_ENABLED` initialization parameter to `TRUE` and AWR snapshot interval greater than 0.

To enable ADDM in a PDB:

1. Set the `AWR_PDB_AUTOFLUSH_ENABLED` initialization parameter to `TRUE` in the PDB using the following command:

```
SQL> ALTER SYSTEM SET AWR_PDB_AUTOFLUSH_ENABLED=TRUE;
```

2. Set the AWR snapshot interval greater than 0 in the PDB using the command as shown in the following example:

```
SQL> EXEC
dbms_workload_repository.modify_snapshot_settings(interval=>60);
```

> **See Also:**
>
> - *Oracle Database Reference* for more information about the `AWR_PDB_AUTOFLUSH_ENABLED` initialization parameter
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS` procedure

Real-Time ADDM Analysis

Introduced in Oracle Enterprise Manager Cloud Control (Cloud Control) 12*c*, Real-Time ADDM helps you to analyze and resolve problems in unresponsive or hung databases that traditionally require you to restart the database. Real-Time ADDM runs through a set of predefined criteria to analyze the current performance of the database. After analyzing the problem, Real-Time ADDM helps you to resolve the identified issues—such as deadlocks, hangs, shared pool contention, and other exception situations—without having to restart the database.

This section describes Real-Time ADDM and contains the following topics:

- Real-Time ADDM Connection Modes
- Real-Time ADDM Triggers
- Real-Time ADDM Trigger Controls

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for information about using Real-Time ADDM with Cloud Control

Real-Time ADDM Connection Modes

Depending on the database state, Real-Time ADDM uses two different types of connection modes when connecting to the database using Cloud Control:

- Normal connection

  In this mode, Real-Time ADDM performs a normal JDBC connection to the database. This mode is intended to perform extensive performance analysis of the database when some connectivity is available.

- Diagnostic connection

  In this mode, Real-Time ADDM performs a latch-less connection to the database. This mode is intended for extreme hang situations when a normal JDBC connection is not possible.

Real-Time ADDM Triggers

Starting with Oracle Database 12*c*, Real-Time ADDM proactively detects transient database performance issues. To do this, Real-Time ADDM runs automatically every 3 seconds and uses in-memory data to diagnose any performance spikes in the database.

Real-Time ADDM triggers an analysis automatically when a performance problem is detected, as described in the following steps:

1. Every 3 seconds, the manageability monitor process (MMON) performs an action to obtain performance statistics without lock or latch.
2. The MMON process checks these statistics and triggers a Real-Time ADDM analysis if any of the issues listed in Table 7-1 are detected.
3. The MMON slave process creates the report and stores it in the AWR.

   To view metadata for the report, use the `DBA_HIST_REPORTS` view.

Table 7-1 lists the issues and conditions that trigger a Real-Time ADDM analysis.

Table 7-1 Triggering Issues and Conditions for Real-Time ADDM

Issue	Condition
High load	Average active sessions are greater than 3 times the number of CPU cores
I/O bound	I/O impact on active sessions based on single block read performance
CPU bound	Active sessions are greater than 10% of total load and CPU utilization is greater than 50%
Over-allocated memory	Memory allocations are over 95% of physical memory
Interconnect bound	Based on single block interconnect transfer time
Session limit	Session limit is close to 100%
Process limit	Process limit is close to 100%
Hung session	Hung sessions are greater than 10% of total sessions
Deadlock detected	Any deadlock is detected

See Also:

Oracle Database Reference for information about the `DBA_HIST_REPORTS` view

Real-Time ADDM Trigger Controls

To ensure that the automatic triggers do not consume too many system resources and overwhelm the system, Real-Time ADDM employs the following controls:

- Duration between reports

If a Real-Time ADDM report was created in the past 5 minutes by the automatic trigger, then no new reports will be generated.

- Oracle RAC control

  Automatic triggers are local to the database instance. For Oracle RAC, only one database instance can create a Real-Time ADDM report at a given time because a lock is required and a query is performed by the MMON slave process before the report is actually generated.

- Repeated triggers

  An automatic trigger for any issue must have an impact of 100% or higher than the previous report with the same triggering issue within the past 45 minutes. For example, if a report is triggered for active sessions with an impact of 8 sessions, then in order for another report to trigger within the next 45 minutes, there must be at least 16 active sessions. In this case, the reported problem with the database is becoming more severe over time. On the other hand, if the same report is being generated once every 45 minutes, then the database is experiencing a persistent problem that has a consistent impact.

- Newly identified issues

  If a new issue is detected (that was not previously detected within the past 45 minutes), then a new report is generated. For example, if a report is triggered for 8 active sessions and a new deadlock issue is detected, then a new report is generated regardless of the new active sessions load.

ADDM Analysis Results

In addition to problem diagnostics, ADDM recommends possible solutions. ADDM analysis results are represented as a set of findings. See Example 7-1 for an example of an ADDM analysis result. Each ADDM finding can belong to one of the following types:

- Problem findings describe the root cause of a database performance problem.
- Symptom findings contain information that often lead to one or more problem findings.
- Information findings are used for reporting information that are relevant to understanding the performance of the database, but do not constitute a performance problem (such as non-problem areas of the database and the activity of automatic database maintenance).
- Warning findings contain information about problems that may affect the completeness or accuracy of the ADDM analysis (such as missing data in AWR).

Each problem finding is quantified by an impact that is an estimate of the portion of `DB time` caused by the finding's performance issue. A problem finding can be associated with a list of recommendations for reducing the impact of the performance problem. The types of recommendations include:

- Hardware changes: adding CPUs or changing the I/O subsystem configuration
- Database configuration: changing initialization parameter settings
- Schema changes: hash partitioning a table or index, or using automatic segment-space management (ASSM)
- Application changes: using the cache option for sequences or using bind variables
- Using other advisors: running SQL Tuning Advisor on high-load SQL or running Segment Advisor on hot objects

A list of recommendations can contain various alternatives for solving the same problem; you do not have to apply all the recommendations to solve a specific problem. Each recommendation has a benefit which is an estimate of the portion of DB time that can be saved if the recommendation is implemented. Recommendations are composed of actions and rationales. You must apply all the actions of a recommendation to gain the estimated benefit. The rationales are used for explaining why the set of actions were recommended and to provide additional information to implement the suggested recommendation.

Reviewing ADDM Analysis Results: Example

Consider the following section of an ADDM report in Example 7-1.

Example 7-1 Example ADDM Report

```
FINDING 1: 31% impact (7798 seconds)
-----------------------------------
SQL statements were not shared due to the usage of literals. This resulted in
additional hard parses which were consuming significant database time.

RECOMMENDATION 1: Application Analysis, 31% benefit (7798 seconds)
  ACTION: Investigate application logic for possible use of bind variables
    instead of literals. Alternatively, you may set the parameter
    "cursor_sharing" to "force".
  RATIONALE: SQL statements with PLAN_HASH_VALUE 3106087033 were found to be
    using literals. Look in V$SQL for examples of such SQL statements.
```

In Example 7-1, the finding points to a particular root cause, the usage of literals in SQL statements, which is estimated to have an impact of about 31% of total DB time in the analysis period.

The finding has a recommendation associated with it, composed of one action and one rationale. The action specifies a solution to the problem found and is estimated to have a maximum benefit of up to 31% DB time in the analysis period. Note that the benefit is given as a portion of the total DB time and not as a portion of the finding's impact. The rationale provides additional information on tracking potential SQL statements that were using literals and causing this performance issue. Using the specified plan hash value of SQL statements that could be a problem, a DBA could quickly examine a few sample statements.

When a specific problem has multiple causes, ADDM may report multiple problem and symptom findings. In this case, the impacts of these multiple findings can contain the same portion of DB time. Because the performance issues of findings can overlap, the sum of the impacts of the findings can exceed 100% of DB time. For example, if a system performs many reads, then ADDM might report a SQL statement responsible for 50% of DB time due to I/O activity as one finding, and an undersized buffer cache responsible for 75% of DB time as another finding.

When multiple recommendations are associated with a problem finding, the recommendations may contain alternatives for solving the problem. In this case, the sum of the recommendations' benefits may be higher than the finding's impact.

When appropriate, an ADDM action may have multiple solutions for you to choose from. In the example, the most effective solution is to use bind variables. However, it is often difficult to modify the application. Changing the value of the CURSOR_SHARING initialization parameter is much easier to implement and can provide significant improvement.

Setting Up ADDM

Automatic database diagnostic monitoring is enabled by default and is controlled by the CONTROL_MANAGEMENT_PACK_ACCESS and the STATISTICS_LEVEL initialization parameters.

The CONTROL_MANAGEMENT_PACK_ACCESS parameter should be set to DIAGNOSTIC or DIAGNOSTIC+TUNING to enable automatic database diagnostic monitoring. The default setting is DIAGNOSTIC+TUNING. Setting CONTROL_MANAGEMENT_PACK_ACCESS to NONE disables ADDM.

The STATISTICS_LEVEL parameter should be set to the TYPICAL or ALL to enable automatic database diagnostic monitoring. The default setting is TYPICAL. Setting STATISTICS_LEVEL to BASIC disables many Oracle Database features, including ADDM, and is strongly discouraged.

> **See Also:**
>
> *Oracle Database Reference* for information about the CONTROL_MANAGEMENT_PACK_ACCESS and STATISTICS_LEVEL initialization parameters

ADDM analysis of I/O performance partially depends on a single argument, DBIO_EXPECTED, that describes the expected performance of the I/O subsystem. The value of DBIO_EXPECTED is the average time it takes to read a single database block in microseconds. Oracle Database uses the default value of 10 milliseconds, which is an appropriate value for most modern hard drives. If your hardware is significantly different—such as very old hardware or very fast RAM disks—then consider using a different value.

To determine the correct setting for the DBIO_EXPECTED parameter:

1. Measure the average read time of a single database block read for your hardware.

   Note that this measurement is for random I/O, which includes seek time if you use standard hard drives. Typical values for hard drives are between 5000 and 20000 microseconds.

2. Set the value one time for all subsequent ADDM executions.

   For example, if the measured value if 8000 microseconds, you should execute the following command as SYS user:

```
EXECUTE DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(
                    'ADDM', 'DBIO_EXPECTED', 8000);
```

Diagnosing Database Performance Problems with ADDM

To diagnose database performance problems, first review the ADDM analysis results that are automatically created each time an AWR snapshot is taken. If a different analysis is required (such as a longer analysis period, using a different DBIO_EXPECTED setting, or changing the analysis mode), you can run ADDM manually as described in this section.

ADDM can analyze any two AWR snapshots (on the same database), as long as both snapshots are still stored in AWR (have not been purged). ADDM can only analyze instances that are started before the beginning snapshot and remain running until the ending snapshot. Additionally, ADDM will not analyze instances that experience significant errors when

generating AWR snapshots. In such cases, ADDM will analyze the largest subset of instances that did not experience these problems.

The primary interface for diagnostic monitoring is Cloud Control. Whenever possible, run ADDM using Cloud Control, as described in *Oracle Database 2 Day + Performance Tuning Guide*. If Cloud Control is unavailable, then run ADDM using the `DBMS_ADDM` package. To run the `DBMS_ADDM` APIs, the user must be granted the `ADVISOR` privilege.

This section contains the following topics:

- Running ADDM in Database Mode
- Running ADDM in Instance Mode
- Running ADDM in Partial Mode
- Displaying an ADDM Report

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_ADDM` package

Running ADDM in Database Mode

For Oracle RAC configurations, you can run ADDM in Database mode to analyze all instances of the databases. For single-instance configurations, you can still run ADDM in Database mode; ADDM will behave as if running in Instance mode.

To run ADDM in Database mode, use the `DBMS_ADDM.ANALYZE_DB` procedure:

```
BEGIN
DBMS_ADDM.ANALYZE_DB (
   task_name           IN OUT VARCHAR2,
   begin_snapshot      IN     NUMBER,
   end_snapshot        IN     NUMBER,
   db_id               IN     NUMBER := NULL);
END;
/
```

The `task_name` parameter specifies the name of the analysis task that will be created. The `begin_snapshot` parameter specifies the snapshot number of the beginning snapshot in the analysis period. The `end_snapshot` parameter specifies the snapshot number of the ending snapshot in the analysis period. The `db_id` parameter specifies the database identifier of the database that will be analyzed. If unspecified, this parameter defaults to the database identifier of the database to which you are currently connected.

The following example creates an ADDM task in database analysis mode, and executes it to diagnose the performance of the entire database during the time period defined by snapshots 137 and 145:

```
VAR tname VARCHAR2(30);
BEGIN
  :tname := 'ADDM for 7PM to 9PM';
  DBMS_ADDM.ANALYZE_DB(:tname, 137, 145);
```

```
END;
/
```

Running ADDM in Instance Mode

To analyze a particular instance of the database, you can run ADDM in Instance mode. To run ADDM in Instance mode, use the `DBMS_ADDM.ANALYZE_INST` procedure:

```
BEGIN
DBMS_ADDM.ANALYZE_INST (
   task_name           IN OUT VARCHAR2,
   begin_snapshot      IN     NUMBER,
   end_snapshot        IN     NUMBER,
   instance_number     IN     NUMBER := NULL,
   db_id               IN     NUMBER := NULL);
END;
/
```

The `task_name` parameter specifies the name of the analysis task that will be created. The `begin_snapshot` parameter specifies the snapshot number of the beginning snapshot in the analysis period. The `end_snapshot` parameter specifies the snapshot number of the ending snapshot in the analysis period. The `instance_number` parameter specifies the instance number of the instance that will be analyzed. If unspecified, this parameter defaults to the instance number of the instance to which you are currently connected. The `db_id` parameter specifies the database identifier of the database that will be analyzed. If unspecified, this parameter defaults to the database identifier of the database to which you are currently connected.

The following example creates an ADDM task in instance analysis mode, and executes it to diagnose the performance of instance number 1 during the time period defined by snapshots 137 and 145:

```
VAR tname VARCHAR2(30);
BEGIN
  :tname := 'my ADDM for 7PM to 9PM';
  DBMS_ADDM.ANALYZE_INST(:tname, 137, 145, 1);
END;
/
```

Running ADDM in Partial Mode

To analyze a subset of all database instances, you can run ADDM in Partial mode. To run ADDM in Partial mode, use the `DBMS_ADDM.ANALYZE_PARTIAL` procedure:

```
BEGIN
DBMS_ADDM.ANALYZE_PARTIAL (
   task_name           IN OUT VARCHAR2,
   instance_numbers    IN     VARCHAR2,
   begin_snapshot      IN     NUMBER,
   end_snapshot        IN     NUMBER,
   db_id               IN     NUMBER := NULL);
END;
/
```

The `task_name` parameter specifies the name of the analysis task that will be created. The `instance_numbers` parameter specifies a comma-delimited list of instance numbers of instances that will be analyzed. The `begin_snapshot` parameter specifies the snapshot number of the beginning snapshot in the analysis period. The `end_snapshot` parameter

specifies the snapshot number of the ending snapshot in the analysis period. The `db_id` parameter specifies the database identifier of the database that will be analyzed. If unspecified, this parameter defaults to the database identifier of the database to which you are currently connected.

The following example creates an ADDM task in partial analysis mode, and executes it to diagnose the performance of instance numbers 1, 2, and 4, during the time period defined by snapshots 137 and 145:

```
VAR tname VARCHAR2(30);
BEGIN
  :tname := 'my ADDM for 7PM to 9PM';
  DBMS_ADDM.ANALYZE_PARTIAL(:tname, '1,2,4', 137, 145);
END;
/
```

Displaying an ADDM Report

To display a text report of an executed ADDM task, use the `DBMS_ADDM.GET_REPORT` function:

```
DBMS_ADDM.GET_REPORT (
   task_name           IN VARCHAR2
  RETURN CLOB);
```

The following example displays a text report of the ADDM task specified by its task name using the `tname` variable:

```
SET LONG 1000000 PAGESIZE 0;
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a `CLOB`, formatted to fit line size of 80. For information about reviewing the ADDM analysis results in an ADDM report, see "ADDM Analysis Results".

ADDM Views

Typically, you should view ADDM analysis using Cloud Control or `DBMS_ADDM` package subprograms.

However, you can also get ADDM information using the `DBA_ADDM_*` and `DBA_ADVISOR_*` views. This group of views includes:

- `DBA_ADVISOR_FINDINGS`

  This view displays all the findings discovered by all advisors. Each finding is displayed with an associated finding ID, name, and type. For tasks with multiple executions, the name of each task execution associated with each finding is also listed.

- `DBA_ADDM_FINDINGS`

  This view contains a subset of the findings displayed in the related `DBA_ADVISOR_FINDINGS` view. This view only displays the ADDM findings discovered by all advisors. Each ADDM finding is displayed with an associated finding ID, name, and type.

- `DBA_ADVISOR_FINDING_NAMES`

This view lists all finding names registered with the advisor framework.

- `DBA_ADVISOR_RECOMMENDATIONS`

  This view displays the results of completed diagnostic tasks with recommendations for the problems identified in each execution. The recommendations should be reviewed in the order of the `RANK` column, as this relays the magnitude of the problem for the recommendation. The `BENEFIT` column displays the benefit to the system you can expect after the recommendation is performed. For tasks with multiple executions, the name of each task execution associated with each advisor task is also listed.

- `DBA_ADVISOR_TASKS`

  This view provides basic information about existing tasks, such as the task ID, task name, and when the task was created. For tasks with multiple executions, the name and type of the last or current execution associated with each advisor task is also listed.

See Also:

- *Oracle Database Reference* for more information about `DBA_ADDM_*` and `DBA_ADVISOR_*` views
- "Displaying an ADDM Report"

8
Comparing Database Performance Over Time

This chapter describes how to compare database performance over time using Automatic Workload Repository (AWR) Compare Periods reports and contains the following topics:

- About Automatic Workload Repository Compare Periods Reports
- Generating Automatic Workload Repository Compare Periods Reports
- Interpreting Automatic Workload Repository Compare Periods Reports

About Automatic Workload Repository Compare Periods Reports

Performance degradation of the database occurs when your database was performing optimally in the past, but has over time gradually degraded to a point where it becomes noticeable to the users. AWR Compare Periods report enables you to compare database performance over time.

An AWR report shows AWR data during a period in time between two snapshots (or two points in time). An AWR Compare Periods report, on the other hand, shows the difference between two periods in time (or two AWR reports, which equates to four snapshots). Using AWR Compare Periods reports helps you to identify detailed performance attributes and configuration settings that differ between two time periods.

For example, assume that a batch workload runs daily during a maintenance window between 10:00 p.m. and midnight is showing poor performance and is now completing at 2 a.m instead. You can generate an AWR Compare Periods report for the time period from 10:00 p.m. to midnight on a day when performance was good, and another report for the time period from 10:00 a.m. to 2 a.m. on a day when performance was poor. You can then compare these reports to identify configuration settings, workload profile, and statistics that differ between these two time periods. Based on those differences, you can more easily diagnose the cause of the performance degradation.

The two time periods selected in an AWR Compare Periods report can be of different durations because the report normalizes the statistics by the amount of time spent on the database for each time period, and presents statistical data ordered by the largest difference between the time periods.

> **Note:**
>
> Data visibility and privilege requirements may differ when using AWR features with pluggable databases (PDBs). For information about how manageability features, including AWR features, work in a multitenant container database (CDB), see *Oracle Multitenant Administrator's Guide*.

> **See Also:**
>
> - "Automatic Workload Repository" for information about the AWR
> - "Generating Automatic Workload Repository Reports" for information about AWR reports

Generating Automatic Workload Repository Compare Periods Reports

If the performance of your database degrades over time, AWR Compare Periods reports enable you to compare two periods in time to identify key differences that can help you diagnose the cause of the performance degradation.

AWR Compare Periods reports are divided into multiple sections. The HTML report includes links that can be used to navigate quickly between sections. The content of the report contains the workload profile of the system for the selected range of snapshots.

- User Interfaces for Generating AWR Compare Periods Reports
- Generating an AWR Compare Periods Report Using the Command-Line Interface

User Interfaces for Generating AWR Compare Periods Reports

The primary interface for generating AWR Compare Periods reports is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, generate AWR Compare Periods reports using Cloud Control.

If Cloud Control is unavailable, then generate AWR Compare Periods reports by running SQL scripts. The DBA role is required to run these scripts.

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for information about generating AWR Compare Periods reports using Cloud Control

Generating an AWR Compare Periods Report Using the Command-Line Interface

This topic describes how to generate AWR Compare Periods reports by running SQL scripts in the command-line interface.

- Generating an AWR Compare Periods Report for the Local Database
- Generating an AWR Compare Periods Report for a Specific Database
- Generating an Oracle RAC AWR Compare Periods Report for the Local Database
- Generating an Oracle RAC AWR Compare Periods Report for a Specific Database

Generating an AWR Compare Periods Report for the Local Database

The `awrddrpt.sql` SQL script generates an HTML or text report that compares detailed performance attributes and configuration settings between two selected time periods on the local database instance.

To generate an AWR Compare Periods report on the local database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/awrddrpt.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: html
   ```

   In this example, an HTML report is chosen.

3. Specify the number of days for which you want to list snapshot IDs in the first time period.

   ```
   Enter value for num_days: 2
   ```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

4. Specify a beginning and ending snapshot ID for the first time period:

   ```
   Enter value for begin_snap: 102
   Enter value for end_snap: 103
   ```

   In this example, the snapshot with a snapshot ID of 102 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 103 is selected as the ending snapshot for the first time period.

5. Specify the number of days for which you want to list snapshot IDs in the second time period.

   ```
   Enter value for num_days2: 1
   ```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

6. Specify a beginning and ending snapshot ID for the second time period:

   ```
   Enter value for begin_snap2: 126
   Enter value for end_snap2: 127
   ```

   In this example, the snapshot with a snapshot ID of 126 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 127 is selected as the ending snapshot for the second time period.

7. Enter a report name, or accept the default report name:

   ```
   Enter value for report_name:
   Using the report name awrdiff_1_102_1_126.txt
   ```

   In this example, the default name is accepted and an AWR report named `awrdiff_1_102_126` is generated.

Generating an AWR Compare Periods Report for a Specific Database

The `awrddrpi.sql` SQL script generates an HTML or text report that compares detailed performance attributes and configuration settings between two selected time periods on a specific database and instance. This script enables you to specify a database identifier and instance for which AWR Compare Periods report will be generated.

To generate an AWR Compare Periods report on a specific database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/awrddrpi.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: text
   ```

   In this example, a text report is chosen.

3. A list of available database identifiers and instance numbers are displayed:

   ```
   Instances in this Workload Repository schema
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      DB Id    Inst Num DB Name      Instance     Host
   ----------- -------- ------------ ------------ ------------
    3309173529        1 MAIN         main         examp1690
    3309173529        1 TINT251      tint251      samp251
   ```

   Enter the values for the database identifier (`dbid`) and instance number (`inst_num`) for the first time period:

   ```
   Enter value for dbid: 3309173529
   Using 3309173529 for Database Id for the first pair of snapshots
   Enter value for inst_num: 1
   Using 1 for Instance Number for the first pair of snapshots
   ```

4. Specify the number of days for which you want to list snapshot IDs in the first time period.

   ```
   Enter value for num_days: 2
   ```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

5. Specify a beginning and ending snapshot ID for the first time period:

   ```
   Enter value for begin_snap: 102
   Enter value for end_snap: 103
   ```

   In this example, the snapshot with a snapshot ID of 102 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 103 is selected as the ending snapshot for the first time period.

6. Enter the values for the database identifier (`dbid`) and instance number (`inst_num`) for the second time period:

   ```
   Enter value for dbid2: 3309173529
   Using 3309173529 for Database Id for the second pair of snapshots
   ```

```
Enter value for inst_num2: 1
Using 1 for Instance Number for the second pair of snapshots
```

7. Specify the number of days for which you want to list snapshot IDs in the second time period.

   ```
   Enter value for num_days2: 1
   ```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

8. Specify a beginning and ending snapshot ID for the second time period:

   ```
   Enter value for begin_snap2: 126
   Enter value for end_snap2: 127
   ```

   In this example, the snapshot with a snapshot ID of 126 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 127 is selected as the ending snapshot for the second time period.

9. Enter a report name, or accept the default report name:

   ```
   Enter value for report_name:
   Using the report name awrdiff_1_102_1_126.txt
   ```

   In this example, the default name is accepted and an AWR report named `awrdiff_1_102_126` is generated on the database instance with a database ID value of `3309173529`.

Generating an Oracle RAC AWR Compare Periods Report for the Local Database

The `awrgdrpt.sql` SQL script generates an HTML or text report that compares detailed performance attributes and configuration settings between two selected time periods using the current database identifier and all available database instances in an Oracle Real Application Clusters (Oracle RAC) environment.

> **Note:**
>
> In an Oracle RAC environment, generate an HTML report (instead of a text report) because it is much easier to read.

To generate an AWR Compare Periods report for Oracle RAC on the local database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/awrgdrpt.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: html
   ```

   In this example, an HTML report is chosen.

3. Specify the number of days for which you want to list snapshot IDs in the first time period.

   ```
   Enter value for num_days: 2
   ```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

4. Specify a beginning and ending snapshot ID for the first time period:

```
Enter value for begin_snap: 102
Enter value for end_snap: 103
```

In this example, the snapshot with a snapshot ID of 102 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 103 is selected as the ending snapshot for the first time period.

5. Specify the number of days for which you want to list snapshot IDs in the second time period.

```
Enter value for num_days2: 1
```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

6. Specify a beginning and ending snapshot ID for the second time period:

```
Enter value for begin_snap2: 126
Enter value for end_snap2: 127
```

In this example, the snapshot with a snapshot ID of 126 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 127 is selected as the ending snapshot for the second time period.

7. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrracdiff_1st_1_2nd_1.html
```

In this example, the default name is accepted and an AWR report named `awrrac_1st_1_2nd_1.html` is generated.

Generating an Oracle RAC AWR Compare Periods Report for a Specific Database

The `awrgdrpi.sql` SQL script generates an HTML or text report that compares detailed performance attributes and configuration settings between two selected time periods using specific databases and instances in an Oracle RAC environment. This script enables you to specify database identifiers and a comma-delimited list of database instances for which AWR Compare Periods report will be generated.

> **Note:**
>
> In an Oracle RAC environment, you should always generate an HTML report (instead of a text report) because they are much easier to read.

To generate an AWR Compare Periods report for Oracle RAC on a specific database using the command-line interface:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/awrgdrpi.sql
```

2. Specify whether you want an HTML or a text report:

```
Enter value for report_type: html
```

   In this example, an HTML report is chosen.

3. A list of available database identifiers and instance numbers are displayed:

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
 3309173529        1 MAIN         main         examp1690
 3309173529        1 TINT251      tint251      samp251
 3309173529        2 TINT251      tint252      samp252
 3309173529        3 TINT251      tint253      samp253
 3309173529        4 TINT251      tint254      samp254
```

   Enter the values for the database identifier (`dbid`) and instance number (`instance_numbers_or_all`) for the first time period:

```
Enter value for dbid: 3309173529
Using 3309173529 for Database Id for the first pair of snapshots
Enter value for inst_num: 1,2
Using instances 1 for the first pair of snapshots
```

4. Specify the number of days for which you want to list snapshot IDs in the first time period.

```
Enter value for num_days: 2
```

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

5. Specify a beginning and ending snapshot ID for the first time period:

```
Enter value for begin_snap: 102
Enter value for end_snap: 103
```

   In this example, the snapshot with a snapshot ID of 102 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 103 is selected as the ending snapshot for the first time period.

6. A list of available database identifiers and instance numbers are displayed:

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
 3309173529        1 MAIN         main         examp1690
 3309173529        1 TINT251      tint251      samp251
 3309173529        2 TINT251      tint252      samp252
 3309173529        3 TINT251      tint253      samp253
 3309173529        4 TINT251      tint254      samp254
INSTNUM1
--------------------------------------------------
1,2
```

   Enter the values for the database identifier (`dbid2`) and instance numbers (`instance_numbers_or_all2`) for the second time period:

```
Enter value for dbid2: 3309173529
Using 3309173529 for Database Id for the second pair of snapshots
Enter value for instance_numbers_or_all2: 3,4
```

7. Specify the number of days for which you want to list snapshot IDs in the second time period.

```
Enter value for num_days2: 1
```

A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

8. Specify a beginning and ending snapshot ID for the second time period:

```
Enter value for begin_snap2: 126
Enter value for end_snap2: 127
```

In this example, the snapshot with a snapshot ID of 126 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 127 is selected as the ending snapshot for the second time period.

9. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name awrracdiff_1st_1_2nd_1.html
```

In this example, the default name is accepted and an AWR report named `awrrac_1st_1_2nd_1.html` is generated.

Interpreting Automatic Workload Repository Compare Periods Reports

After generating an AWR Compare Periods report for the time periods you want to compare, review its contents to identify possible causes of performance degradation over time.

The content of the AWR Compare Periods report is divided into the following sections:

- Summary of the AWR Compare Periods Report
- Details of the AWR Compare Periods Report
- Supplemental Information in the AWR Compare Periods Report

Summary of the AWR Compare Periods Report

The report summary is at the beginning of the AWR Compare Periods report, and summarizes information about the snapshot sets and workloads used in the report.

The report summary contains the following sections:

- Snapshot Sets
- Host Configuration Comparison
- System Configuration Comparison
- Load Profile
- Top 5 Timed Events

Snapshot Sets

The Snapshot Sets section displays information about the snapshot sets used for this report, such as instance, host, and snapshot information.

Host Configuration Comparison

The Host Configuration Comparison section compares the host configurations used in the two snapshot sets. For example, the report compares physical memory and number of CPUs. Any differences in the configurations are quantified as percentages differed in the %Diff column.

System Configuration Comparison

The System Configuration Comparison section compares the database configurations used in the two snapshot sets. For example, the report compares the System Global Area (SGA) and log buffer sizes. Any differences in the configurations are quantified as percentages differed in the %Diff column.

Load Profile

The Load Profile section compares the workloads used in the two snapshot sets. Any differences in the workloads are quantified as percentages differed in the %Diff column.

Top 5 Timed Events

The Top 5 Timed Events section displays the five timed events or operations that consumed the highest percentage of total database time (DB time) in each of the snapshot sets.

Details of the AWR Compare Periods Report

The details section follows the report summary of the AWR Compare Periods report, and provides extensive information about the snapshot sets and workloads used in the report.

The report details contains the following sections:

- Time Model Statistics
- Operating System Statistics
- Wait Events
- Service Statistics
- SQL Statistics
- Instance Activity Statistics
- I/O Statistics
- Advisory Statistics
- Wait Statistics
- Undo Segment Summary
- Latch Statistics
- Segment Statistics

- In-Memory Segment Statistics
- Dictionary Cache Statistics
- Library Cache Statistics
- Memory Statistics
- Streams Statistics

Time Model Statistics

The Time Model Statistics section compares time model statistics in the two snapshot sets. The time model statistics are ordered based on the difference in total DB time spent on a particular type of operation between the two snapshot sets, and are listed in descending order. Time model statistics at the top of this section have the greatest differential between the two snapshot sets, and the related operations may be possible causes for performance degradation over time.

> **See Also:**
>
> "Time Model Statistics" for information about time model statistics

Operating System Statistics

The Operating System Statistics section compares operating system statistics in the two snapshot sets. This section provides an overall state of the operating system during each of the two periods being compared.

Wait Events

The Wait Events section compares the wait events in the two snapshot sets.

The first section lists the classes of wait events, including user I/O and system I/O. The classes are listed in descending order by absolute value of the % of DB time column.

The second section lists the wait events. The wait events are ordered based on the difference in total DB time spent on the wait event between the two snapshot sets, and are listed in descending order. Wait events at the top of this section have the greatest differential between the two snapshot sets, and may be possible causes for performance degradation over time.

> **See Also:**
>
> "Wait Events Statistics" for information about wait events and wait classes

Service Statistics

The Service Statistics section compares services in the two snapshot sets. The services are ordered based on the difference in total DB time spent on a particular service between the two snapshot sets, and are listed in descending order.

SQL Statistics

The SQL Statistics section compares the top SQL statements in the two snapshot sets. The SQL statements are ordered based on different comparison methods, but in all cases, the top ten SQL statements with the greatest differential between the two snapshot sets are shown.

The SQL statements shown in this section may be possible causes for performance degradation over time, and are ordered based on the following categories:

- Top 10 SQL Comparison by Execution Time
- Top 10 SQL Comparison by CPU Time
- Top 10 SQL Comparison by Buffer Gets
- Top 10 SQL Comparison by Physical Reads
- Top 10 SQL Comparison by Executions
- Top 10 SQL Comparison by Parse Calls
- Complete List of SQL Text

Top 10 SQL Comparison by Execution Time

SQL statements in this subsection are ordered based on the difference in total DB time spent processing the SQL statement between the two snapshot sets and are listed in descending order.

SQL statements shown in this subsection that consumed a high percentage of DB time in the one time period, but not in the other, are likely the high-load SQL statements that caused the performance degradation and should be investigated. Review the SQL statements in the Complete List of SQL Text subsection of the report and tune them, if necessary.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for information about tuning SQL statements

Top 10 SQL Comparison by CPU Time

SQL statements in this subsection are ordered based on the difference in CPU time spent processing the SQL statement between the two snapshot sets, and are listed in descending order.

Top 10 SQL Comparison by Buffer Gets

SQL statements in this subsection are ordered based on the difference in the number of total buffer cache reads or buffer gets made when processing the SQL statement between the two snapshot sets, and are listed in descending order.

Top 10 SQL Comparison by Physical Reads

SQL statements in this subsection are ordered based on the difference in the number of physical reads made when processing the SQL statement between the two snapshot sets, and are listed in descending order.

Top 10 SQL Comparison by Executions

SQL statements in this subsection are ordered based on the difference in the number of executions per second (based on DB time) when processing the SQL statement between the two snapshot sets, and are listed in descending order.

Top 10 SQL Comparison by Parse Calls

SQL statements in this subsection are ordered based on the difference in the number of total parses made when processing the SQL statement between the two snapshot sets, and are listed in descending order. Parsing is one stage in the processing of a SQL statement.

When an application issues a SQL statement, the application makes a parse call to Oracle Database. Making parse calls can greatly affect the performance of a database and should be minimized as much as possible.

> **See Also:**
>
> *Oracle Database Concepts* for information about parsing

Complete List of SQL Text

This subsection displays the SQL text of all SQL statements listed in the SQL Statistics section.

Instance Activity Statistics

The Instance Activity Statistics section compares the statistic values of instance activity between the two snapshot sets. For each statistic, the value of the statistic is shown along with the differentials measured by DB time, elapsed time, and per transaction.

The instance activity statistics are categorized into the following subsections:

- Key Instance Activity Statistics
- Other Instance Activity Statistics

Key Instance Activity Statistics

This subsection displays the difference in key instance activity statistic values between the two snapshot sets.

Other Instance Activity Statistics

This subsection displays the difference in instance activity for all other statistics between the two snapshot sets.

I/O Statistics

The I/O Statistics section compares the I/O operations performed on tablespaces and database files between the two snapshot sets. A drastic increase in I/O operations between the two snapshots may be the cause of performance degradation over time.

For each tablespace or database file, the difference in the number of reads, writes, and buffer cache waits (or buffer gets) are quantified as a percentage. The database files are ordered based on different comparison methods, but in all cases, the top 10 database files with the greatest differential between the two snapshot sets are shown.

The I/O statistics are divided into the following categories:

- Tablespace I/O Statistics
- Top 10 File Comparison by I/O
- Top 10 File Comparison by Read Time
- Top 10 File Comparison by Buffer Waits

Tablespace I/O Statistics

Tablespaces shown in this subsection are ordered by the difference in the number of normalized I/Os performed on the tablespace between the two snapshot sets, and are listed in descending order. Normalized I/Os are the sum of average reads and writes per second.

Top 10 File Comparison by I/O

Database files shown in this subsection are ordered by the difference in the number of normalized I/Os performed on the database file between the two snapshot sets, and are listed in descending order. Normalized I/Os are the sum of average reads and writes per second.

Top 10 File Comparison by Read Time

Database files shown in this subsection are ordered by the difference in the percentage of DB time spent reading data from the database file between the two snapshot sets, and are listed in descending order.

Top 10 File Comparison by Buffer Waits

Database files shown in this subsection are ordered by the difference in the number of buffer waits (waits caused during a free buffer lookup in the buffer cache) performed on the database file between the two snapshot sets, and are listed in descending order.

Advisory Statistics

The Advisory Statistics section compares program global area (PGA) memory statistics between the two snapshot sets, and is divided into the following categories:

- PGA Aggregate Summary
- PGA Aggregate Target Statistics

PGA Aggregate Summary

This subsection compares the PGA cache hit ratio between the two snapshot sets.

PGA Aggregate Target Statistics

This subsection compares the key statistics related to the automatic PGA memory management between the two snapshot sets.

Wait Statistics

The Wait Statistics section compares statistics for buffer waits and enqueues between the two snapshot sets.

The wait statistics are divided into the following categories:

- Buffer Wait Statistics
- Enqueue Activity

Buffer Wait Statistics

This subsection compares buffer waits between the two snapshot sets. Buffer waits happen during a free buffer lookup in the buffer cache.

Enqueue Activity

This subsection compares enqueue activities between the two snapshot sets. Enqueues are shared memory structures (or locks) that serialize access to database resources and can be associated with a session or transaction.

> **See Also:**
>
> *Oracle Database Reference* for information about enqueues

Undo Segment Summary

The Undo Segment Summary section compares the use of undo segments in the two periods. The chart compares the number of undo blocks in the two periods, the number of transactions that use those blocks, and the maximum length of queries. The STO/OOS column indicates the number of snapshot too old and out of space counts.

Latch Statistics

The Latch Statistics section compares the number of total sleeps for latches between the two snapshot sets in descending order.

Latches are simple, low-level serialization mechanisms to protect shared data structures in the SGA. For example, latches protect the list of users currently accessing the database and the data structures describing the blocks in the buffer cache. A server or background process acquires a latch for a very short time while manipulating or looking up one of these structures. The implementation of latches is

operating system dependent, particularly in regard to whether and how long a process will wait for a latch.

Segment Statistics

The Segment Statistics section compares segments, or database objects (such as tables and indexes), between the two snapshot sets. The segments are ordered based on different comparison methods, but in all cases the top five segments with the greatest differential between the two snapshot sets are shown.

The segments shown in this may be the causes of performance degradation over time, and are ordered based on the following categories:

- Top 5 Segments Comparison by Logical Reads
- Top 5 Segments Comparison by Physical Reads
- Top 5 Segments Comparison by Row Lock Waits
- Top 5 Segments Comparison by ITL Waits
- Top 5 Segments Comparison by Buffer Busy Waits

Top 5 Segments Comparison by Logical Reads

Segments shown in this subsection are ordered based on the difference in the number of logical reads (total number of reads from disk or memory) performed on the segment between the two snapshot sets, and are listed in descending order.

If an extremely high percentage of logical reads are made on a database object, then the associated SQL statements should be investigated to determine if data access to the database object need to be tuned using an index or a materialized view.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for information about optimizing data access paths

Top 5 Segments Comparison by Physical Reads

Segments shown in this subsection are ordered based on the difference in the number of physical reads (such as disk reads) performed on the segment between the two snapshot sets, and are listed in descending order.

Top 5 Segments Comparison by Row Lock Waits

Segments shown in this subsection are ordered based on the difference in the number of waits on row locks for the segment between the two snapshot sets, and are listed in descending order.

Row-level locks are primarily used to prevent two transactions from modifying the same row. When a transaction needs to modify a row, a row lock is acquired.

> **See Also:**
>
> *Oracle Database Concepts* for information about row locks

Top 5 Segments Comparison by ITL Waits
: Segments shown in this subsection are ordered based on the difference in the number of interested transaction list (ITL) waits for the segment between the two snapshot sets, and are listed in descending order.

Top 5 Segments Comparison by Buffer Busy Waits
: Segments shown in this subsection are ordered based on the difference in the number of buffer busy waits for the segment between the two snapshot sets, and are listed in descending order.

In-Memory Segment Statistics

The In-Memory Segment Statistics section compares in-memory segment statistics between the two snapshot sets and lists the top in-memory segments based on number of scans, database block changes, populate CU activities, and repopulate CU activities. These statistics provide an insight into how in-memory segments are utilized by user workload. The In-Memory Segment Statistics section is displayed in AWR Compare Periods report only if Oracle Database has in-memory activity.

Dictionary Cache Statistics

The Dictionary Cache Statistics section compares the number of get requests performed on the dictionary cache between the two snapshot sets in descending order. The difference is measured by the number of get requests per second of both total DB time and elapsed time.

The dictionary cache is a part of the SGA that stores information about the database, its structures, and its users. The dictionary cache also stores descriptive information (or metadata) about schema objects, which is accessed by Oracle Database during the parsing of SQL statements.

> **See Also:**
>
> "Data Dictionary Cache Concepts" for information about the dictionary cache

Library Cache Statistics

The Library Cache Statistics section compares the number of get requests performed on the library cache between the two snapshot sets in descending order. The difference is measured by the number of get requests per second of both total DB time and elapsed time.

The library cache is a part of the SGA that stores table information, object definitions, SQL statements, and PL/SQL programs.

> **See Also:**
>
> "Library Cache Concepts" for information about the library cache

Memory Statistics

The Memory Statistics section compares process and SGA memory statistics between the two snapshot sets, and is divided into the following categories:

- Process Memory Summary
- SGA Memory Summary
- SGA Breakdown Difference

Process Memory Summary

This subsection summarizes the memory use of processes in the two time periods. The process categories include SQL, PL/SQL, and other.

SGA Memory Summary

This subsection summarizes the SGA memory configurations for the two snapshot sets.

SGA Breakdown Difference

This subsection compares SGA memory usage for each of its subcomponents between the two snapshot sets. The difference is measured based on the percentage changed in the beginning and ending values of memory usage between the two snapshot sets.

Advanced Queuing Statistics

The Advanced Queuing Statistics section compares CPU time, I/O time, and other statistics.

Supplemental Information in the AWR Compare Periods Report

The supplemental information is at the end of the AWR Compare Periods report, and provides information that is useful but not essential about the snapshot sets and workloads used in the report.

The supplemental information contains the following sections:

- init.ora Parameters
- Complete List of SQL Text

init.ora Parameters

The init.ora Parameters section lists all the initialization parameter values for the first snapshot set. Any changes in the values of the initialization parameters between the two snapshot sets are listed for the second snapshot set with the changed value shown.

Complete List of SQL Text

The Complete List of SQL Text section lists each statement contained in the workloads by SQL ID and shows the text of the SQL statement.

9
Analyzing Sampled Data

This chapter describes how to use sampled data to identify transient performance problems in Oracle Database and contains the following topics:

About Active Session History

The Active Session History (ASH) is a diagnostic tool that records the information about all the active sessions in an Oracle database.

The Automatic Database Diagnostics Monitor (ADDM) analysis may not show transient performance problems because they are short-lived. The ASH diagnostic tool captures transient performance problems by taking samples of active sessions every second and storing the sampled data in a circular buffer in the shared global area (SGA). Any session that is connected to the database and is waiting for an event that does not belong to the Idle wait class is considered as an active session. By capturing only active sessions, a manageable set of data is represented with its size being directly related to the work being performed, rather than the number of sessions allowed on the system.

ASH enables you to examine and perform detailed analysis on the sampled session activity using the `V$ACTIVE_SESSION_HISTORY` view. The data present in ASH can be rolled up in various dimensions that it captures over a specified duration and gathered into an ASH report.

Note:

ADDM tries to report the most significant performance problems during an analysis period in terms of their impact on DB time. Whether a performance problem is captured by ADDM depends on its duration compared to the interval between AWR snapshots.

If a performance problem lasts for a significant portion of the time between snapshots, it will be captured by ADDM. For example, if the snapshot interval is set to one hour, then a performance problem that lasts for 30 minutes should not be considered as a transient performance problem because its duration represents a significant portion of the snapshot interval and will likely be captured by ADDM.

If a particular problem lasts for a very short duration, then its severity might be averaged out or minimized by other performance problems in the analysis period, and the problem may not appear in the ADDM findings. Using the same example where the snapshot interval is set to one hour, a performance problem that lasts for only 2 minutes may be a transient performance problem because its duration represents a small portion of the snapshot interval and will likely not show up in the ADDM findings.

See Also:

- "Active Session History Statistics" for information about ASH
- *Oracle Multitenant Administrator's Guide* for information about how manageability features, such as ASH, work in a multitenant container database.

Generating Active Session History Reports

ASH reports enable you to perform analysis of:

- Transient performance problems that typically last for a few minutes
- Scoped or targeted performance analysis by various dimensions or their combinations, such as time, session, module, action, or SQL identifier

ASH reports are divided into multiple sections. The HTML report includes links that can be used to navigate quickly between sections. The content of the report contains ASH information used to identify blocker and waiter identities, their associated transaction identifiers, and SQL statements for a specified duration.

This section describes how to generate ASH reports and contains the following topics:

- User Interfaces for Generating ASH Reports
- Generating an ASH Report Using the Command-Line Interface

User Interfaces for Generating ASH Reports

The primary interface for generating ASH reports is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, generate ASH reports using Cloud Control.

If Cloud Control is unavailable, then generate ASH reports by running SQL scripts. The DBA role is required to run these scripts.

> **See Also:**
>
> *Oracle Database 2 Day + Performance Tuning Guide* for information about generating ASH reports using Cloud Control

Generating an ASH Report Using the Command-Line Interface

This section describes how to generate ASH reports by running SQL scripts in the command-line interface.

This section contains the following topics:

- Generating an ASH Report on the Local Database Instance
- Generating an ASH Report on a Specific Database Instance
- Generating an ASH Report for Oracle RAC

Generating an ASH Report on the Local Database Instance

The `ashrpt.sql` SQL script generates an HTML or text report that displays ASH information for a specified duration on the local database instance.

To generate an ASH report on the local database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/ashrpt.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: text
   ```

   In this example, a text report is chosen.

3. Specify the begin time in minutes before the system date:

   ```
   Enter value for begin_time: -10
   ```

   In this example, 10 minutes before the current time is selected.

4. Specify the duration to capture ASH information in minutes from the begin time.

   ```
   Enter value for duration:
   ```

   In this example, the default duration of system date minus begin time is accepted.

5. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name ashrpt_1_0310_0131.txt
```

In this example, the default name is accepted and an ASH report named `ashrpt_1_0310_0131` is generated. The report will gather ASH information beginning from 10 minutes before the current time and ending at the current time.

Generating an ASH Report on a Specific Database Instance

The `ashrpti.sql` SQL script generates an HTML or text report that displays ASH information for a specified duration on a specified database and instance. This script enables you to specify a database and instance for which the ASH report will be generated.

To generate an ASH report on a specific database instance using the command-line interface:

1. At the SQL prompt, enter:

   ```
   @$ORACLE_HOME/rdbms/admin/ashrpti.sql
   ```

2. Specify whether you want an HTML or a text report:

   ```
   Enter value for report_type: html
   ```

   In this example, an HTML report is chosen.

   A list of available database IDs and instance numbers are displayed:

   ```
   Instances in this Workload Repository schema
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      DB Id    Inst Num DB Name      Instance     Host
   ----------- -------- ------------ ------------ ------------
    3309173529        1 MAIN         main         examp1690
    3309173529        1 TINT251      tint251      samp251
   ```

3. Enter the values for the database identifier (`dbid`) and instance number (`inst_num`):

   ```
   Enter value for dbid: 3309173529
   Using 3309173529 for database id
   Enter value for inst_num: 1
   ```

4. To generate an ASH report on a physical standby instance, the standby database must be opened read-only. The ASH data on disk represents activity on the primary database and the ASH data in memory represents activity on the standby database.

   > **Note:**
   >
   > This step is applicable only if you are generating an ASH report on an Active Data Guard physical standby instance. If this is not the case, then skip this step.

   Specify whether to generate the report using data sampled from the primary or standby database:

```
You are running ASH report on a Standby database.
To generate the report over data sampled on the Primary database, enter 'P'.
Defaults to 'S' - data sampled in the Standby database.
Enter value for stdbyflag:
Using Primary (P) or Standby (S): S
```

In this example, the default value of Standby (S) is selected.

5. Specify the begin time in minutes before the system date:

```
Enter value for begin_time: -10
```

In this example, 10 minutes before the current time is selected.

6. Specify the duration to capture ASH information in minutes from the begin time.

```
Enter value for duration:
```

In this example, the default duration of system date minus begin time is accepted.

7. Specify the slot width in seconds that will be used in the Activity Over Time section of the report:

```
Enter value for slot_width:
```

In this example, the default value is accepted. For more information about the Activity Over Time section and how to specify the slot width, see "Activity Over Time".

8. Follow the instructions in the subsequent prompts and enter values for the following report targets:
   - `target_session_id`
   - `target_sql_id`
   - `target_wait_class`
   - `target_service_hash`
   - `target_module_name`
   - `target_action_name`
   - `target_client_id`
   - `target_plsql_entry`
9. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name ashrpt_1_0310_0131.txt
```

In this example, the default name is accepted and an ASH report named `ashrpt_1_0310_0131` is generated. The report will gather ASH information on the database instance with a database ID value of `3309173529` beginning from 10 minutes before the current time and ending at the current time.

Generating an ASH Report for Oracle RAC

The `ashrpti.sql` SQL script generates an HTML or text report that displays ASH information for a specified duration for specified databases and instances in an Oracle Real Application Clusters (Oracle RAC) environment. Only ASH data that is written to disk will be used to generate the report. This report will only use ASH samples from the last 10 minutes that are found in the `DBA_HIST_ACTIVE_SESS_HISTORY` table.

To generate an ASH report for Oracle RAC:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/ashrpti.sql
```

2. Specify whether you want an HTML or a text report:

```
Enter value for report_type: html
```

In this example, an HTML report is chosen.

A list of available database IDs and instance numbers are displayed:

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
------------ -------- ------------ ------------ ------------
 3309173529        1 MAIN         main         examp1690
 3309173529        1 TINT251      tint251      samp251
 3309173529        2 TINT251      tint252      samp252
 3309173529        3 TINT251      tint253      samp253
 3309173529        4 TINT251      tint254      samp254
```

3. Enter the values for the database identifier (`dbid`) and instance number (`inst_num`):

```
Enter value for dbid: 3309173529
Using database id: 3309173529
Enter instance numbers. Enter 'ALL' for all instances in an Oracle
RAC cluster or explicitly specify list of instances (e.g., 1,2,3).
Defaults to current instance.
Enter value for inst_num: ALL
Using instance number(s): ALL
```

4. Specify the begin time in minutes before the system date:

```
Enter value for begin_time: -1:10
```

In this example, 1 hour and 10 minutes before the current time is selected.

5. Specify the duration to capture ASH information in minutes from the begin time.

```
Enter value for duration: 10
```

In this example, the duration is set to 10 minutes.

6. Specify the slot width in seconds that will be used in the Activity Over Time section of the report:

```
Enter value for slot_width:
```

In this example, the default value is accepted. For more information about the Activity Over Time section and how to specify the slot width, see "Activity Over Time".

7. Follow the instructions in the subsequent prompts and enter values for the following report targets:

- `target_session_id`
- `target_sql_id`
- `target_wait_class`

- target_service_hash
- target_module_name
- target_action_name
- target_client_id
- target_plsql_entry

8. Enter a report name, or accept the default report name:

```
Enter value for report_name:
Using the report name ashrpt_rac_0310_0131.txt
```

In this example, the default name is accepted and an ASH report named `ashrpt_rac_0310_0131` is generated. The report will gather ASH information on all instances belonging to the database with a database ID value of `3309173529` beginning from 1 hour and 10 minutes before the current time and ending at 1 hour before the current time.

Interpreting Results from Active Session History Reports

After generating an ASH report, review its contents to identify possible causes of transient performance problems.

The contents of the ASH report are divided into the following sections:

- Top Events
- Load Profile
- Top SQL
- Top PL/SQL
- Top Java
- Top Sessions
- Top Objects/Files/Latches
- Activity Over Time

> **See Also:**
>
> *Oracle Real Application Clusters Administration and Deployment Guide* for information about sections in the ASH report that are specific to Oracle Real Application Clusters (Oracle RAC)

Top Events

The Top Events section describes the top wait events of the sampled session activity categorized by user, background, and priority. Use the information in this section to identify wait events that may be causing a transient performance problem.

The Top Events section contains the following subsections:

- Top User Events

- Top Background Events
- Top Event P1/P2/P3

Top User Events

The Top User Events subsection lists the top wait events from user processes that accounted for the highest percentages of sampled session activity.

Top Background Events

The Top Background Events subsection lists the top wait events from backgrounds that accounted for the highest percentages of sampled session activity.

Top Event P1/P2/P3

The Top Event P1/P2/P3 subsection lists the wait event parameter values of the top wait events that accounted for the highest percentages of sampled session activity, ordered by the percentage of total wait time (% Event). For each wait event, values in the P1 Value, P2 Value, P3 Value column correspond to wait event parameters displayed in the Parameter 1, Parameter 2, and Parameter 3 columns.

Load Profile

The Load Profile section describes the load analyzed in the sampled session activity. Use the information in this section to identify the service, client, or SQL command type that may be the cause of a transient performance problem.

The Load Profile section contains the following subsections:

- Top Service/Module
- Top Client IDs
- Top SQL Command Types
- Top Phases of Execution

Top Service/Module

The Top Service/Module subsection lists the services and modules that accounted for the highest percentages of sampled session activity.

Top Client IDs

The Top Client IDs subsection lists the clients that accounted for the highest percentages of sampled session activity based on their client ID, which is the application-specific identifier of the database session.

Top SQL Command Types

The Top SQL Command Types subsection lists the SQL command types—such as `SELECT` or `UPDATE` commands—that accounted for the highest percentages of sampled session activity.

Top Phases of Execution

The Top Phases of Execution subsection lists the phases of execution—such as SQL, PL/SQL, and Java compilation and execution—that accounted for the highest percentages of sampled session activity.

Top SQL

The Top SQL section describes the top SQL statements in the sampled session activity. Use this information to identify high-load SQL statements that may be the cause of a transient performance problem.

The Top SQL section contains the following subsections:

- Top SQL with Top Events
- Top SQL with Top Row Sources
- Top SQL Using Literals
- Top Parsing Module/Action
- Complete List of SQL Text

Top SQL with Top Events

The Top SQL with Top Events subsection lists the SQL statements that accounted for the highest percentages of sampled session activity and the top wait events that were encountered by these SQL statements. The Sampled # of Executions column shows how many distinct executions of a particular SQL statement were sampled.

Top SQL with Top Row Sources

The Top SQL with Top Row Sources subsection lists the SQL statements that accounted for the highest percentages of sampled session activity and their detailed execution plan information. You can use this information to identify which part of the SQL execution contributed significantly to the SQL elapsed time.

Top SQL Using Literals

The Top SQL Using Literals subsection lists the SQL statements using literals that accounted for the highest percentages of sampled session activity. You should review the statements listed in this report to determine whether the literals can be replaced with bind variables.

Top Parsing Module/Action

The Top Parsing Module/Action subsection lists the module and action that accounted for the highest percentages of sampled session activity while parsing the SQL statement.

Complete List of SQL Text

The Complete List of SQL Text subsection displays the entire text of the SQL statements shown in the Top SQL section.

Top PL/SQL

The Top PL/SQL section lists the PL/SQL procedures that accounted for the highest percentages of sampled session activity.

The PL/SQL Entry Subprogram column lists the application's top-level entry point into PL/SQL. The PL/SQL Current Subprogram column lists the PL/SQL subprogram being executed at the point of sampling. If the value of this column is `SQL`, then the % Current column shows the percentage of time spent executing SQL for this subprogram.

Top Java

The Top Java section describes the top Java programs in the sampled session activity.

Top Sessions

The Top Sessions section describes the sessions that were waiting for a particular wait event. Use this information to identify the sessions that accounted for the highest percentages of sampled session activity, which may be the cause of a transient performance problem.

The Top Sessions section contains the following subsections:

- Top Sessions
- Top Blocking Sessions
- Top Sessions Running PQs

Top Sessions

The Top Session subsection lists the sessions that were waiting for a particular wait event that accounted for the highest percentages of sampled session activity.

Top Blocking Sessions

The Top Blocking Sessions subsection lists the blocking sessions that accounted for the highest percentages of sampled session activity.

Top Sessions Running PQs

The Top Sessions Running PQs subsection lists the sessions running parallel queries (PQs) that were waiting for a particular wait event, which accounted for the highest percentages of sampled session activity.

Top Objects/Files/Latches

The Top Objects/Files/Latches section provides additional information about the most commonly-used database resources and contains the following subsections:

- Top DB Objects
- Top DB Files
- Top Latches

Top DB Objects

The Top DB Objects subsection lists the database objects (such as tables and indexes) that accounted for the highest percentages of sampled session activity.

Top DB Files

The Top DB Files subsection lists the database files that accounted for the highest percentages of sampled session activity.

Top Latches

The Top Latches subsection lists the latches that accounted for the highest percentages of sampled session activity.

Latches are simple, low-level serialization mechanisms used to protect shared data structures in the System Global Area (SGA). For example, latches protect the list of users currently accessing the database and the data structures describing the blocks in the buffer cache. A server or background process acquires a latch for a very short time while manipulating or looking at one of these structures. The implementation of latches is operating system-dependent, particularly regarding if and how long a process waits for a latch.

Activity Over Time

The Activity Over Time section is one of the most informative sections of the ASH report. This section is particularly useful for analyzing longer time periods because it provides in-depth details about activities and workload profiles during the analysis period.

The Activity Over Time section is divided into 10 time slots. The size of each time slot varies based on the duration of the analysis period. The first and last slots are usually odd-sized. All inner slots are equally sized and can be compared to each other. For example, if the analysis period lasts for 10 minutes, then all time slots will 1 minute each. However, if the analysis period lasts for 9 minutes and 30 seconds, then the outer slots may be 15 seconds each and the inner slots will be 1 minute each.

Each of the time slots contains information regarding that particular time slot, as described in Table 9-1.

Table 9-1 Activity Over Time

Column	Description
Slot Time (Duration)	Duration of the slot
Slot Count	Number of sampled sessions in the slot
Event	Top three wait events in the slot
Event Count	Number of ASH samples waiting for the wait event
% Event	Percentage of ASH samples waiting for wait events in the entire analysis period

When comparing the inner slots, perform a skew analysis by identifying spikes in the Event Count and Slot Count columns. A spike in the Event Count column indicates an increase in the number of sampled sessions waiting for a particular event. A spike in the Slot Count column indicates an increase in active sessions, because ASH data is sampled from active

sessions only and a relative increase in database workload. Typically, when the number of active session samples and the number of sessions associated with a wait event increases, the slot may be the cause of a transient performance problem.

To generate the ASH report with a user-defined slot size, run the `ashrpti.sql` script, as described in "Generating an ASH Report on a Specific Database Instance".

10

Instance Tuning Using Performance Views

After the initial configuration of a database, monitoring and tuning an instance regularly is important to eliminate any potential performance bottlenecks. This chapter discusses the tuning process using Oracle `V$` performance views.

This chapter contains the following sections:

Instance Tuning Steps

These are the main steps in the Oracle performance method for instance tuning:

1. Define the Problem

   Get candid feedback from users about the scope of the performance problem.

2. Examine the Host System and Examine the Oracle Database Statistics
   - After obtaining a full set of operating system, database, and application statistics, examine the data for any evidence of performance problems.
   - Consider the list of common performance errors to see whether the data gathered suggests that they are contributing to the problem.
   - Build a conceptual model of what is happening on the system using the performance data gathered.

3. Implement and Measure Change

   Propose changes to be made and the expected result of implementing the changes. Then, implement the changes and measure application performance.

4. Determine whether the performance objective defined in step 1 has been met. If not, then repeat steps 2 and 3 until the performance goals are met.

The remainder of this chapter discusses instance tuning using the Oracle Database dynamic performance views. However, Oracle recommends using Automatic Workload Repository (AWR) and Automatic Database Diagnostic Monitor (ADDM) for statistics gathering, monitoring, and tuning due to the extended feature list.

> **Note:**
>
> If your site does not have AWR and ADDM features, then you can use Statspack to gather Oracle database instance statistics.

Define the Problem

It is vital to develop a good understanding of the purpose of the tuning exercise and the nature of the problem before attempting to implement a solution. Without this understanding, it is virtually impossible to implement effective changes. The data gathered during this stage helps determine the next step to take and what evidence to examine.

Gather the following data:

1. Identify the performance objective.

   What is the measure of acceptable performance? How many transactions an hour, or seconds, response time will meet the required performance level?

2. Identify the scope of the problem.

   What is affected by the slowdown? For example, is the whole instance slow? Is it a particular application, program, specific operation, or a single user?

3. Identify the time frame when the problem occurs.

   Is the problem only evident during peak hours? Does performance deteriorate over the course of the day? Was the slowdown gradual (over the space of months or weeks) or sudden?

4. Quantify the slowdown.

   This helps identify the extent of the problem and also acts as a measure for comparison when deciding whether changes implemented to fix the problem have actually made an improvement. Find a consistently reproducible measure of the response time or job run time. How much worse are the timings than when the program was running well?

5. Identify any changes.

   Identify what has changed since performance was acceptable. This may narrow the potential cause quickly. For example, has the operating system software, hardware, application software, or Oracle Database release been upgraded? Has more data been loaded into the system, or has the data volume or user population grown?

At the end of this phase, you should have a good understanding of the symptoms. If the symptoms can be identified as local to a program or set of programs, then the problem is handled in a different manner from instance-wide performance issues.

Examine the Host System

Look at the load on the database server and the database instance. Consider the operating system, the I/O subsystem, and network statistics, because examining these areas helps determine what might be worth further investigation. In multitier systems, also examine the application server middle-tier hosts.

Examining the host hardware often gives a strong indication of the bottleneck in the system. This determines which Oracle Database performance data could be useful for cross-reference and further diagnosis.

Data to examine includes the following:

- CPU Usage

- Identifying I/O Problems
- Identifying Network Issues

CPU Usage

If there is a significant amount of idle CPU, then there could be an I/O, application, or database bottleneck. Note that wait I/O should be considered as idle CPU.

If there is high CPU usage, then determine whether the CPU is being used effectively. Is the majority of CPU usage attributable to a small number of high-CPU using programs, or is the CPU consumed by an evenly distributed workload?

If a small number of high-usage programs use the CPU, then look at the programs to determine the cause. Check whether some processes alone consume the full power of one CPU. Depending on the process, this could indicate a CPU or process-bound workload that can be tackled by dividing or parallelizing process activity.

Non-Oracle Processes

If the programs are not Oracle programs, then identify whether they are legitimately requiring that amount of CPU. If so, determine whether their execution be delayed to off-peak hours. Identifying these CPU intensive processes can also help narrowing what specific activity, such as I/O, network, and paging, is consuming resources and how can it be related to the database workload.

Oracle Processes

If a small number of Oracle processes consumes most of the CPU resources, then use `SQL_TRACE` and `TKPROF` to identify the SQL or PL/SQL statements to see if a particular query or PL/SQL program unit can be tuned. For example, a SELECT statement could be CPU-intensive if its execution involves many reads of data in cache (logical reads) that could be avoided with better SQL optimization.

Oracle Database CPU Statistics

Oracle Database CPU statistics are available in several `V$` views:

- `V$SYSSTAT` shows Oracle Database CPU usage for all sessions. The `CPU used by this session` statistic shows the aggregate CPU used by all sessions. The `parse time cpu` statistic shows the total CPU time used for parsing.
- `V$SESSTAT` shows Oracle Database CPU usage for each session. Use this view to determine which particular session is using the most CPU.
- `V$RSRC_CONSUMER_GROUP` shows CPU utilization statistics for each consumer group when the Oracle Database Resource Manager is running.

Interpreting CPU Statistics

It is important to recognize that CPU time and real time are distinct. With eight CPUs, for any given minute in real time, there are eight minutes of CPU time available. On Windows and UNIX, this can be either user time or system time (privileged mode on Windows). Thus, average CPU time utilized by all processes (threads) on the system could be greater than one minute for every one minute real time interval.

At any given moment, you know how much time Oracle Database has used on the system. So, if eight minutes are available and Oracle Database uses four minutes of that time, then you know that 50% of all CPU time is used by Oracle. If your process is not consuming that time, then some other process is. Identify the processes that are using CPU time, figure out why, and then attempt to tune them.

If the CPU usage is evenly distributed over many Oracle server processes, examine the `V$SYS_TIME_MODEL` view to help get a precise understanding of where most time is spent.

> **See Also:**
>
> "Table 10-1" for more information about various wait events and their possible causes

Identifying I/O Problems

An overly active I/O system can be evidenced by disk queue lengths greater than two, or disk service times that are over 20-30ms. If the I/O system is overly active, then check for potential hot spots that could benefit from distributing the I/O across more disks. Also identify whether the load can be reduced by lowering the resource requirements of the programs using those resources. If the I/O problems are caused by Oracle Database, then I/O tuning can begin. If Oracle Database is not consuming the available I/O resources, then identify the process that is using up the I/O. Determine why the process is using up the I/O, and then tune this process.

I/O problems can be identified using `V$` views in Oracle Database and monitoring tools in the operating system, as described in the following sections:

- Identifying I/O Problems Using V$ Views
- Identifying I/O Problems Using Operating System Monitoring Tools

Identifying I/O Problems Using V$ Views

Check the Oracle wait event data in `V$SYSTEM_EVENT` to see whether the top wait events are I/O related. I/O related events include `db file sequential read`, `db file scattered read`, `db file single write`, `db file parallel write`, and `log file parallel write`. These are all events corresponding to I/Os performed against data files and log files. If any of these wait events correspond to high average time, then investigate the I/O contention.

Cross reference the host I/O system data with the I/O sections in the Automatic Repository report to identify hot data files and tablespaces. Also compare the I/O times reported by the operating system with the times reported by Oracle Database to see if they are consistent.

An I/O problem can also manifest itself with non-I/O related wait events. For example, the difficulty in finding a free buffer in the buffer cache or high wait times for logs to be flushed to disk can also be symptoms of an I/O problem. Before investigating whether the I/O system should be reconfigured, determine if the load on the I/O system can be reduced.

To reduce I/O load caused by Oracle Database, examine the I/O statistics collected for all I/O calls made by the database using the following views:

- V$IOSTAT_CONSUMER_GROUP

  The V$IOSTAT_CONSUMER_GROUP view captures I/O statistics for consumer groups. If Oracle Database Resource Manager is enabled, I/O statistics for all consumer groups that are part of the currently enabled resource plan are captured.

- V$IOSTAT_FILE

  The V$IOSTAT_FILE view captures I/O statistics of database files that are or have been accessed. The SMALL_SYNC_READ_LATENCY column displays the latency for single block synchronous reads (in milliseconds), which translates directly to the amount of time that clients need to wait before moving onto the next operation. This defines the responsiveness of the storage subsystem based on the current load. If there is a high latency for critical data files, you may want to consider relocating these files to improve their service time. To calculate latency statistics, timed_statistics must be set to TRUE.

- V$IOSTAT_FUNCTION

  The V$IOSTAT_FUNCTION view captures I/O statistics for database functions (such as the LGWR and DBWR).

  An I/O can be issued by various Oracle processes with different functionalities. The top database functions are classified in the V$IOSTAT_FUNCTION view. In cases when there is a conflict of I/O functions, the I/O is placed in the bucket with the lower FUNCTION_ID. For example, if XDB issues an I/O from the buffer cache, the I/O would be classified as an XDB I/O because it has a lower FUNCTION_ID value. Any unclassified function is placed in the Others bucket. You can display the FUNCTION_ID hierarchy by querying the V$IOSTAT_FUNCTION view:

```
select FUNCTION_ID, FUNCTION_NAME
from v$iostat_function
order by FUNCTION_ID;

FUNCTION_ID FUNCTION_NAME
----------- ------------------
          0 RMAN
          1 DBWR
          2 LGWR
          3 ARCH
          4 XDB
          5 Streams AQ
          6 Data Pump
          7 Recovery
          8 Buffer Cache Reads
          9 Direct Reads
         10 Direct Writes
         11 Others
```

These V$IOSTAT views contains I/O statistics for both single and multi block read and write operations. Single block operations are small I/Os that are less than or equal to 128 kilobytes. Multi block operations are large I/Os that are greater than 128 kilobytes. For each of these operations, the following statistics are collected:

- Identifier
- Total wait time (in milliseconds)
- Number of waits executed (for consumer groups and functions)

- Number of requests for each operation
- Number of single and multi block bytes read
- Number of single and multi block bytes written

You should also look at SQL statements that perform many physical reads by querying the `V$SQLAREA` view, or by reviewing the "SQL ordered by Reads" section of the Automatic Workload Repository report. Examine these statements to see how they can be tuned to reduce the number of I/Os.

> **See Also:**
>
> *Oracle Database Reference* for more information about the views `V$IOSTAT_CONSUMER_GROUP`, `V$IOSTAT_FUNCTION`, `V$IOSTAT_FILE`, and `V$SQLAREA`

Identifying I/O Problems Using Operating System Monitoring Tools

Use operating system monitoring tools to determine what processes are running on the system as a whole and to monitor disk access to all files. Remember that disks holding data files and redo log files can also hold files that are not related to Oracle Database. Reduce any heavy access to disks that contain database files. You can monitor access to non-database files only through operating system facilities, rather than through the `V$` views.

Utilities, such as `sar -d` (or `iostat`) on many UNIX systems and the administrative performance monitoring tool on Windows systems, examine I/O statistics for the entire system.

> **See Also:**
>
> Your operating system documentation for the tools available on your platform

Identifying Network Issues

Using operating system utilities, look at the network round-trip ping time and the number of collisions. If the network is causing large delays in response time, then investigate possible causes.

To identify network I/O caused by remote access of database files, examine the `V$IOSTAT_NETWORK` view. This view contains network I/O statistics caused by accessing files on a remote database instance, including:

- Database client initiating the network I/O (such as RMAN and PLSQL)
- Number of read and write operations issued
- Number of kilobytes read and written
- Total wait time in milliseconds for read operations
- Total wait in milliseconds for write operations

After the cause of the network issue is identified, network load can be reduced by scheduling large data transfers to off-peak times, or by coding applications to batch requests to remote hosts, rather than accessing remote hosts once (or more) for one request.

Examine the Oracle Database Statistics

Examine Oracle Database statistics and cross-reference them with operating system statistics to ensure a consistent diagnosis of the problem. Operating system statistics can indicate a good place to begin tuning. However, if the goal is to tune the Oracle database instance, then look at the Oracle Database statistics to identify the resource bottleneck from a database perspective before implementing corrective action.

This section contains the following topics.

- Setting the Level of Statistics Collection
- Wait Events
- Dynamic Performance Views Containing Wait Event Statistics
- System Statistics
- Segment-Level Statistics

> **See Also:**
>
> "Interpreting Oracle Database Statistics"

Setting the Level of Statistics Collection

Oracle Database provides the initialization parameter `STATISTICS_LEVEL`, which controls all major statistics collections or advisories in the database. This parameter sets the statistics collection level for the database.

Depending on the setting of `STATISTICS_LEVEL`, certain advisories or statistics are collected, as follows:

- `BASIC`: No advisories or statistics are collected. Monitoring and many automatic features are disabled. Oracle does not recommend this setting because it disables important Oracle Database features.
- `TYPICAL`: This is the default value and ensures collection for all major statistics while providing best overall database performance. This setting should be adequate for most environments.
- `ALL`: All of the advisories or statistics that are collected with the `TYPICAL` setting are included, plus timed operating system statistics and row source execution statistics.

See Also:

- *Oracle Database Reference* for more information on the `STATISTICS_LEVEL` initialization parameter.
- *Oracle Database Reference* for information about the `V$STATISTICS_LEVEL` view. This view lists the status of the statistics or advisories controlled by the `STATISTICS_LEVEL` initialization parameter.

Wait Events

Wait events are statistics that are incremented by a server process or thread to indicate that it had to wait for an event to complete before being able to continue processing. Wait event data reveals various symptoms of problems that might be impacting performance, such as latch contention, buffer contention, and I/O contention. Remember that these are only symptoms of problems, not the actual causes.

Wait events are grouped into classes. The wait event classes include: Administrative, Application, Cluster, Commit, Concurrency, Configuration, Idle, Network, Other, Scheduler, System I/O, and User I/O.

A server process can wait for the following:

- A resource to become available, such as a buffer or a latch.
- An action to complete, such as an I/O.
- More work to do, such as waiting for the client to provide the next SQL statement to execute. Events that identify that a server process is waiting for more work are known as idle events.

Wait event statistics include the number of times an event was waited for and the time waited for the event to complete. If the initialization parameter `TIMED_STATISTICS` is set to `true`, then you can also see how long each resource was waited for.

To minimize user response time, reduce the time spent by server processes waiting for event completion. Not all wait events have the same wait time. Therefore, it is more important to examine events with the most total time waited rather than wait events with a high number of occurrences. Usually, it is best to set the dynamic parameter `TIMED_STATISTICS` to `true` at least while monitoring performance.

See Also:

- "Wait Events Statistics"
- "Using Wait Events with Timed Statistics"
- *Oracle Database Reference* for more information about Oracle Database wait events

Dynamic Performance Views Containing Wait Event Statistics

These dynamic performance views can be queried for wait event statistics:

- `V$ACTIVE_SESSION_HISTORY`

  The `V$ACTIVE_SESSION_HISTORY` view displays active database session activity, sampled once every second.

- `V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL`

  The `V$SESS_TIME_MODEL` and `V$SYS_TIME_MODEL` views contain time model statistics, including `DB time` which is the total time spent in database calls.

- `V$SESSION_WAIT`

  The `V$SESSION_WAIT` view displays information about the current or last wait for each session (such as wait ID, class, and time).

- `V$SESSION`

  The `V$SESSION` view displays information about each current session and contains the same wait statistics as those found in the `V$SESSION_WAIT` view. If applicable, this view also contains detailed information about the object that the session is currently waiting for (such as object number, block number, file number, and row number), the blocking session responsible for the current wait (such as the blocking session ID, status, and type), and the amount of time waited.

- `V$SESSION_EVENT`

  The `V$SESSION_EVENT` view provides summary of all the events the session has waited for since it started.

- `V$SESSION_WAIT_CLASS`

  The `V$SESSION_WAIT_CLASS` view provides the number of waits and the time spent in each class of wait events for each session.

- `V$SESSION_WAIT_HISTORY`

  The `V$SESSION_WAIT_HISTORY` view displays information about the last ten wait events for each active session (such as event type and wait time).

- `V$SYSTEM_EVENT`

  The `V$SYSTEM_EVENT` view provides a summary of all the event waits on the instance since it started.

- `V$EVENT_HISTOGRAM`

  The `V$EVENT_HISTOGRAM` view displays a histogram of the number of waits, the maximum wait, and total wait time on an event basis.

- `V$FILE_HISTOGRAM`

  The `V$FILE_HISTOGRAM` view displays a histogram of times waited during single block reads for each file.

- `V$SYSTEM_WAIT_CLASS`

  The `V$SYSTEM_WAIT_CLASS` view provides the instance wide time totals for the number of waits and the time spent in each class of wait events.

- `V$TEMP_HISTOGRAM`

The V$TEMP_HISTOGRAM view displays a histogram of times waited during single block reads for each temporary file.

Investigate wait events and related timing data when performing reactive performance tuning. The events with the most time listed against them are often strong indications of the performance bottleneck. For example, by looking at V$SYSTEM_EVENT, you might notice lots of buffer busy waits. It might be that many processes are inserting into the same block and must wait for each other before they can insert. The solution could be to use automatic segment space management or partitioning for the object in question.

See Also:

- "Wait Events Statistics" for differences among the views V$SESSION_WAIT, V$SESSION_EVENT, and V$SYSTEM_EVENT
- *Oracle Database Reference* for information about the dynamic performance views

System Statistics

System statistics are typically used in conjunction with wait event data to find further evidence of the cause of a performance problem.

For example, if V$SYSTEM_EVENT indicates that the largest wait event (in terms of wait time) is the event buffer busy waits, then look at the specific buffer wait statistics available in the view V$WAITSTAT to see which block type has the highest wait count and the highest wait time.

After the block type has been identified, also look at V$SESSION real-time while the problem is occurring or V$ACTIVE_SESSION_HISTORY and DBA_HIST_ACTIVE_SESS_HISTORY views after the problem has been experienced to identify the contended-for objects using the object number indicated. The combination of this data indicates the appropriate corrective action.

Statistics are available in many V$ views. The following are some of the V$ views that contain system statistics.

V$ACTIVE_SESSION_HISTORY

This view displays active database session activity, sampled once every second.

V$SYSSTAT

This contains overall statistics for many different parts of Oracle Database, including rollback, logical and physical I/O, and parse data. Data from V$SYSSTAT is used to compute ratios, such as the buffer cache hit ratio.

V$FILESTAT

This contains detailed file I/O statistics for each file, including the number of I/Os for each file and the average read time.

V$ROLLSTAT

This contains detailed rollback and undo segment statistics for each segment.

V$ENQUEUE_STAT

This contains detailed enqueue statistics for each enqueue, including the number of times an enqueue was requested and the number of times an enqueue was waited for, and the wait time.

V$LATCH

This contains detailed latch usage statistics for each latch, including the number of times each latch was requested and the number of times the latch was waited for.

> **See Also:**
>
> *Oracle Database Reference* for information about dynamic performance views

Segment-Level Statistics

You can gather segment-level statistics to help you spot performance problems associated with individual segments. Collecting and viewing segment-level statistics is a good way to effectively identify hot tables or indexes in an instance.

After viewing wait events and system statistics to identify the performance problem, you can use segment-level statistics to find specific tables or indexes that are causing the problem. Consider, for example, that `V$SYSTEM_EVENT` indicates that buffer busy waits cause a fair amount of wait time. You can select from `V$SEGMENT_STATISTICS` the top segments that cause the buffer busy waits. Then you can focus your effort on eliminating the problem in those segments.

You can query segment-level statistics through the following dynamic performance views:

- `V$SEGSTAT_NAME`: This view lists the segment statistics being collected and the properties of each statistic (for instance, if it is a sampled statistic).
- `V$SEGSTAT`: This is a highly efficient, real-time monitoring view that shows the statistic value, statistic name, and other basic information.
- `V$SEGMENT_STATISTICS`: This is a user-friendly view of statistic values. In addition to all the columns of `V$SEGSTAT`, it has information about such things as the segment owner and table space name. It makes the statistics easy to understand, but it is more costly.

> **See Also:**
>
> *Oracle Database Reference* for information about dynamic performance views

Implement and Measure Change

Often at the end of a tuning exercise, it is possible to identify two or three changes that could potentially alleviate the problem. To identify which change provides the most benefit, it is recommended that only one change be implemented at a time. The effect of the change should be measured against the baseline data measurements found in the problem definition phase.

Typically, most sites with dire performance problems implement several overlapping changes at once, and thus cannot identify which changes provided any benefit. Although this is not immediately an issue, this becomes a significant hindrance if similar problems subsequently appear, because it is not possible to know which of the changes provided the most benefit and which efforts to prioritize.

If it is not possible to implement changes separately, then try to measure the effects of dissimilar changes. For example, measure the effect of making an initialization change to optimize redo generation separately from the effect of creating a new index to improve the performance of a modified query. It is impossible to measure the benefit of performing an operating system upgrade if SQL is tuned, the operating system disk layout is changed, and the initialization parameters are also changed at the same time.

Performance tuning is an iterative process. It is unlikely to find a 'silver bullet' that solves an instance-wide performance problem. In most cases, excellent performance requires iteration through the performance tuning phases, because solving one bottleneck often uncovers another (sometimes worse) problem.

Knowing when to stop tuning is also important. The best measure of performance is user perception, rather than how close the statistic is to an ideal value.

Interpreting Oracle Database Statistics

Gather statistics that cover the time when the instance had the performance problem. If you previously captured baseline data for comparison, then you can compare the current data to the data from the baseline that most represents the problem workload.

When comparing two reports, ensure that the two reports are from times where the system was running comparable workloads.

Examine Load

Usually, wait events are the first data examined. However, if you have a baseline report, then check to see if the load has changed. Regardless of whether you have a baseline, it is useful to see whether the resource usage rates are high.

Load-related statistics to examine include `redo size`, `session logical reads`, `db block changes`, `physical reads`, `physical read total bytes`, `physical writes`, `physical write total bytes`, `parse count` (`total`), `parse count` (`hard`), and `user calls`. This data is queried from `V$SYSSTAT`. It is best to normalize this data over seconds and over transactions. It is also useful to examine the total I/O load in MB per second by using the sum of physical read total bytes and physical write total bytes. The combined value includes the I/O's used to buffer cache, redo logs, archive logs, by Recovery Manager (RMAN) backup and recovery and any Oracle Database background process.

In the AWR report, look at the Load Profile section. The data has been normalized over transactions and over seconds.

Changing Load

The load profile statistics over seconds show the changes in throughput (that is, whether the instance is performing more work each second). The statistics over transactions identify changes in the application characteristics by comparing these to the corresponding statistics from the baseline report.

High Rates of Activity

Examine the statistics normalized over seconds to identify whether the rates of activity are very high. It is difficult to make blanket recommendations on high values, because the thresholds are different on each site and are contingent on the application characteristics, the number and speed of CPUs, the operating system, the I/O system, and the Oracle Database release.

The following are some generalized examples (acceptable values vary at each site):

- A hard parse rate of more than 100 a second indicates that there is a very high amount of hard parsing on the system. High hard parse rates cause serious performance issues and must be investigated. Usually, a high hard parse rate is accompanied by latch contention on the shared pool and library cache latches.
- Check whether the sum of the wait times for library cache and shared pool latch events (latch: library cache, latch: library cache pin, latch: library cache lock and latch: shared pool) is significant compared to statistic `DB time` found in `V$SYSSTAT`. If so, examine the `SQL ordered by Parse Calls` section of the AWR report.
- A high soft parse rate could be in the rate of 300 a second or more. Unnecessary soft parses also limit application scalability. Optimally, a SQL statement should be soft parsed once in each session and executed many times.

Using Wait Event Statistics to Drill Down to Bottlenecks

Whenever an Oracle process waits for something, it records the wait using one of a set of predefined wait events. These wait events are grouped in wait classes. The Idle wait class groups all events that a process waits for when it does not have work to do and is waiting for more work to perform. Non-idle events indicate nonproductive time spent waiting for a resource or action to complete.

> **Note:**
>
> Not all symptoms can be evidenced by wait events. See "Additional Statistics" for the statistics that can be checked.

The most effective way to use wait event data is to order the events by the wait time. This is only possible if `TIMED_STATISTICS` is set to `true`. Otherwise, the wait events can only be ranked by the number of times waited, which is often not the ordering that best represents the problem.

To get an indication of where time is spent, follow these steps:

1. Examine the data collection for `V$SYSTEM_EVENT`. The events of interest should be ranked by wait time.

   Identify the wait events that have the most significant percentage of wait time. To determine the percentage of wait time, add the total wait time for all wait events, excluding idle events, such as `Null event` , `SQL*Net message from client`, `SQL*Net message to client`, and `SQL*Net more data to client`. Calculate the relative percentage of the five most prominent events by dividing each event's wait time by the total time waited for all events.

   Alternatively, look at the Top 5 Timed Events section at the beginning of the Automatic Workload Repository report. This section automatically orders the wait events (omitting idle events), and calculates the relative percentage:

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                          % Total
Event                                       Waits    Time (s) Call Time
----------------------------------------- ------------ ----------- ---------
CPU time                                                       559     88.80
log file parallel write                        2,181          28      4.42
SQL*Net more data from client                516,611          27      4.24
db file parallel write                        13,383          13      2.04
db file sequential read                          563           2       .27
```

   In some situations, there might be a few events with similar percentages. This can provide extra evidence if all the events are related to the same type of resource request (for example, all I/O related events).

2. Look at the number of waits for these events, and the average wait time. For example, for I/O related events, the average time might help identify whether the I/O system is slow. The following example of this data is taken from the Wait Event section of the AWR report:

```
                                                          Avg
                                                         wait     Waits
Event                            Waits  Timeouts   Time (s)    (ms)      /txn
-------------------------- ---------- ---------- ---------- ------ ---------
log file parallel write         2,181          0         28     13      41.2
SQL*Net more data from clie   516,611          0         27      0   9,747.4
db file parallel write         13,383          0         13      1     252.5
```

3. The top wait events identify the next places to investigate. A table of common wait events is listed in Table 10-1. It is usually a good idea to also have quick look at high-load SQL.

4. Examine the related data indicated by the wait events to see what other information this data provides. Determine whether this information is consistent with the wait event data. In most situations, there is enough data to begin developing a theory about the potential causes of the performance bottleneck.

5. To determine whether this theory is valid, cross-check data you have examined with other statistics available for consistency. The appropriate statistics vary depending on the problem, but usually include load profile-related data in `V$SYSSTAT`, operating system statistics, and so on. Perform cross-checks with other data to confirm or refute the developing theory.

> **See Also:**
>
> - "Idle Wait Events" for the list of idle wait events
> - *Oracle Database Reference* for more information about wait events

Table of Wait Events and Potential Causes

Table 10-1 links wait events to possible causes and gives an overview of the Oracle data that could be most useful to review next.

Table 10-1 Wait Events and Potential Causes

Wait Event	General Area	Possible Causes	Look for / Examine
`buffer busy waits`	Buffer cache, DBWR	Depends on buffer type. For example, waits for an index block may be caused by a primary key that is based on an ascending sequence.	Examine `V$SESSION` while the problem is occurring to determine the type of block in contention.
`free buffer waits`	Buffer cache, DBWR, I/O	Slow DBWR (possibly due to I/O?) Cache too small	Examine write time using operating system statistics. Check buffer cache statistics for evidence of too small cache.
`db file scattered read`	I/O, SQL statement tuning	Poorly tuned SQL Slow I/O system	Investigate `V$SQLAREA` to see whether there are SQL statements performing many disk reads. Cross-check I/O system and `V$FILESTAT` for poor read time.
`db file sequential read`	I/O, SQL statement tuning	Poorly tuned SQL Slow I/O system	Investigate `V$SQLAREA` to see whether there are SQL statements performing many disk reads. Cross-check I/O system and `V$FILESTAT` for poor read time.
`enqueue` waits (waits starting with `enq:`)	Locks	Depends on type of enqueue	Look at `V$ENQUEUE_STAT`.
library cache latch waits: `library cache`, `library cache pin`, and `library cache lock`	Latch contention	SQL parsing or sharing	Check `V$SQLAREA` to see whether there are SQL statements with a relatively high number of parse calls or a high number of child cursors (column `VERSION_COUNT`). Check parse statistics in `V$SYSSTAT` and their corresponding rate for each second.
`log buffer space`	Log buffer, I/O	Log buffer small Slow I/O system	Check the statistic `redo buffer allocation retries` in `V$SYSSTAT`. Check configuring log buffer section in configuring memory chapter. Check the disks that house the online redo logs for resource contention.
`log file sync`	I/O, over- committing	Slow disks that store the online logs Un-batched commits	Check the disks that house the online redo logs for resource contention. Check the number of transactions (`commits` + `rollbacks`) each second, from `V$SYSSTAT`.

> **See Also:**
>
> - "Wait Events Statistics" for detailed information on each event listed in "Table 10-1" and for other information to cross-check
> - *Oracle Database Reference* for information about dynamic performance views
> - My Oracle Support notices on `buffer busy waits` (34405.1) and `free buffer waits` (62172.1). You can also access these notices and related notices by searching for "busy buffer waits" and "free buffer waits" on My Oracle Support website.

Additional Statistics

There are several statistics that can indicate performance problems that do not have corresponding wait events.

Redo Log Space Requests Statistic

The `V$SYSSTAT` statistic `redo log space requests` indicates how many times a server process had to wait for space in the online redo log, not for space in the redo log buffer. Use this statistic and the wait events as an indication that you must tune checkpoints, DBWR, or archiver activity, not LGWR. Increasing the size of the log buffer does not help.

Read Consistency

Your system might spend excessive time rolling back changes to blocks in order to maintain a consistent view. Consider the following scenarios:

- If there are many small transactions and an active long-running query is running in the background on the same table where the changes are happening, then the query might need to roll back those changes often, in order to obtain a read-consistent image of the table. Compare the following `V$SYSSTAT` statistics to determine whether this is happening:
  - `consistent changes` statistic indicates the number of times a database block has rollback entries applied to perform a consistent read on the block. Workloads that produce a great deal of `consistent changes` can consume a great deal of resources.
  - `consistent gets`: statistic counts the number of logical reads in consistent mode.
- If there are few very, large rollback segments, then your system could be spending a lot of time rolling back the transaction table during delayed block cleanout in order to find out exactly which system change number (SCN) a transaction was committed. When Oracle Database commits a transaction, all modified blocks are not necessarily updated with the commit SCN immediately. In this case, it is done later on demand when the block is read or updated. This is called delayed block cleanout.

  The ratio of the following `V$SYSSTAT` statistics should be close to one:

```
ratio = transaction tables consistent reads - undo records applied /
        transaction tables consistent read rollbacks
```

The recommended solution is to use automatic undo management.

- If there are insufficient rollback segments, then there is rollback segment (header or block) contention. Evidence of this problem is available by the following:
  - Comparing the number of `WAITS` to the number of `GETS` in `V$ROLLSTAT`; the proportion of `WAITS` to `GETS` should be small.
  - Examining `V$WAITSTAT` to see whether there are many `WAITS` for buffers of `CLASS` '`undo header`'.

  The recommended solution is to use automatic undo management.

Table Fetch by Continued Row

You can detect migrated or chained rows by checking the number of `table fetch continued row` statistic in `V$SYSSTAT`. A small number of chained rows (less than 1%) is unlikely to impact system performance. However, a large percentage of chained rows can affect performance.

Chaining on rows larger than the block size is inevitable. Consider using a tablespace with a larger block size for such data.

However, for smaller rows, you can avoid chaining by using sensible space parameters and good application design. For example, do *not* insert a row with key values filled in and nulls in most other columns, then update that row with the real data, causing the row to grow in size. Rather, insert rows filled with data from the start.

If an `UPDATE` statement increases the amount of data in a row so that the row no longer fits in its data block, then Oracle Database tries to find another block with enough free space to hold the entire row. If such a block is available, then Oracle Database moves the entire row to the new block. This operation is called **row migration**. If the row is too large to fit into any available block, then the database splits the row into multiple pieces and stores each piece in a separate block. This operation is called **row chaining**. The database can also chain rows when they are inserted.

Migration and chaining are especially detrimental to performance with the following:

- `UPDATE` statements that cause migration and chaining to perform poorly
- Queries that select migrated or chained rows because these must perform additional input and output

The definition of a sample output table named `CHAINED_ROWS` appears in a SQL script available on your distribution medium. The common name of this script is `UTLCHN1.SQL`, although its exact name and location varies depending on your platform. Your output table must have the same column names, data types, and sizes as the `CHAINED_ROWS` table.

Increasing `PCTFREE` can help to avoid migrated rows. If you leave more free space available in the block, then the row has room to grow. You can also reorganize or re-create tables and indexes that have high deletion rates. If tables frequently have rows deleted, then data blocks can have partially free space in them. If rows are inserted and later expanded, then the inserted rows might land in blocks with deleted rows but still not have enough room to expand. Reorganizing the table ensures that the main free space is totally empty blocks.

> **Note:**
>
> PCTUSED is not the opposite of PCTFREE.

> **See Also:**
>
> - *Oracle Database Concepts* for more information on PCTUSED
> - *Oracle Database Administrator's Guide* to learn how to reorganize tables

Parse-Related Statistics

The more your application parses, the more potential for contention exists, and the more time your system spends waiting. If parse time CPU represents a large percentage of the CPU time, then time is being spent parsing instead of executing statements. If this is the case, then it is likely that the application is using literal SQL and so SQL cannot be shared, or the shared pool is poorly configured.

There are several statistics available to identify the extent of time spent parsing by Oracle. Query the parse related statistics from V$SYSSTAT. For example:

```
SELECT NAME, VALUE
  FROM V$SYSSTAT
 WHERE NAME IN (  'parse time cpu', 'parse time elapsed',
                  'parse count (hard)', 'CPU used by this session' );
```

There are various ratios that can be computed to assist in determining whether parsing may be a problem:

- parse time CPU / parse time elapsed

  This ratio indicates how much of the time spent parsing was due to the parse operation itself, rather than waiting for resources, such as latches. A ratio of one is good, indicating that the elapsed time was not spent waiting for highly contended resources.

- parse time CPU / CPU used by this session

  This ratio indicates how much of the total CPU used by Oracle server processes was spent on parse-related operations. A ratio closer to zero is good, indicating that the majority of CPU is not spent on parsing.

Wait Events Statistics

The V$SESSION, V$SESSION_WAIT, V$SESSION_HISTORY, V$SESSION_EVENT, and V$SYSTEM_EVENT views provide information on what resources were waited for, and, if the configuration parameter TIMED_STATISTICS is set to true, how long each resource was waited for.

> **See Also:**
>
> - "Setting the Level of Statistics Collection" for information about the `STATISTICS_LEVEL` settings
> - *Oracle Database Reference* for information about the `V$` views containing wait event statistics

Investigate wait events and related timing data when performing reactive performance tuning. The events with the most time listed against them are often strong indications of the performance bottleneck.

The following views contain related, but different, views of the same data:

- `V$SESSION` lists session information for each current session. It lists either the event currently being waited for, or the event last waited for on each session. This view also contains information about blocking sessions, the wait state, and the wait time.
- `V$SESSION_WAIT` is a current state view. It lists either the event currently being waited for, or the event last waited for on each session, the wait state, and the wait time.
- `V$SESSION_WAIT_HISTORY` lists the last 10 wait events for each current session and the associated wait time.
- `V$SESSION_EVENT` lists the cumulative history of events waited for on each session. After a session exits, the wait event statistics for that session are removed from this view.
- `V$SYSTEM_EVENT` lists the events and times waited for by the whole instance (that is, all session wait events data rolled up) since instance startup.

Because `V$SESSION_WAIT` is a current state view, it also contains a finer-granularity of information than `V$SESSION_EVENT` or `V$SYSTEM_EVENT`. It includes additional identifying data for the current event in three parameter columns: `P1`, `P2`, and `P3`.

For example, `V$SESSION_EVENT` can show that session 124 (SID=124) had many waits on the `db file scattered read`, but it does not show which file and block number. However, `V$SESSION_WAIT` shows the file number in `P1`, the block number read in `P2`, and the number of blocks read in `P3` (`P1` and `P2` let you determine for which segments the wait event is occurring).

This section concentrates on examples using `V$SESSION_WAIT`. However, Oracle recommends capturing performance data over an interval and keeping this data for performance and capacity analysis. This form of rollup data is queried from the `V$SYSTEM_EVENT` view by AWR.

Most commonly encountered events are described in this chapter, listed in case-sensitive alphabetical order. Other event-related data to examine is also included. The case used for each event name is that which appears in the `V$SYSTEM_EVENT` view.

Changes to Wait Event Statistics from Past Releases

Starting with Oracle Database 11*g*, Oracle Database accumulates wait counts and time outs for wait events (such as in the `V$SYSTEM_EVENT` view) differently than in past releases. Continuous waits for certain types of resources (such as enqueues) are internally divided into a set of shorter wait calls. In releases prior to Oracle Database 11*g*, each individual internal

wait call was counted as a separate wait. Starting with Oracle Database 11*g*, a single resource wait is recorded as a single wait, irrespective of the number of internal time outs experienced by the session during the wait.

This change allows Oracle Database to display a more representative wait count, and an accurate total time spent waiting for the resource. Time outs now refer to the resource wait, instead of the individual internal wait calls. This change also affects the average wait time and the maximum wait time. For example, if a user session must wait for an enqueue in order for a transaction row lock to update a single row in a table, and it takes 10 seconds to acquire the enqueue, Oracle Database breaks down the enqueue wait into 3-second wait calls. In this example, there will be three 3-second wait calls, followed by a 1-second wait call. From the session's perspective, however, there is only one wait on an enqueue.

In releases prior to Oracle Database 11*g*, the `V$SYSTEM_EVENT` view would represent this wait scenario as follows:

- `TOTAL_WAITS`: 4 waits (three 3-second waits, one 1-second wait)
- `TOTAL_TIMEOUTS`: 3 time outs (the first three waits time out and the enqueue is acquired during the final wait)
- `TIME_WAITED`: 10 seconds (sum of the times from the 4 waits)
- `AVERAGE_WAIT`: 2.5 seconds
- `MAX_WAIT`: 3 seconds

Starting with Oracle Database 11*g*, this wait scenario is represented as:

- `TOTAL_WAITS`: 1 wait (one 10-second wait)
- `TOTAL_TIMEOUTS`: 0 time outs (the enqueue is acquired during the resource wait)
- TIME_WAITED: 10 seconds (time for the resource wait)
- `AVERAGE_WAIT`: 10 seconds
- `MAX_WAIT`: 10 seconds

The following common wait events are affected by this change:

- Enqueue waits (such as `enq: name - reason` waits)
- Library cache lock waits
- Library cache pin waits
- Row cache lock waits

The following statistics are affected by this change:

- Wait counts
- Wait time outs
- Average wait time
- Maximum wait time

The following views are affected by this change:

- `V$EVENT_HISTOGRAM`
- `V$EVENTMETRIC`

- V$SERVICE_EVENT
- V$SERVICE_WAIT_CLASS
- V$SESSION_EVENT
- V$SESSION_WAIT
- V$SESSION_WAIT_CLASS
- V$SESSION_WAIT_HISTORY
- V$SYSTEM_EVENT
- V$SYSTEM_WAIT_CLASS
- V$WAITCLASSMETRIC
- V$WAITCLASSMETRIC_HISTORY

> **See Also:**
>
> *Oracle Database Reference* for a description of the V$SYSTEM_EVENT view

buffer busy waits

This wait indicates that there are some buffers in the buffer cache that multiple processes are attempting to access concurrently. Query V$WAITSTAT for the wait statistics for each class of buffer. Common buffer classes that have buffer busy waits include data block, segment header, undo header, and undo block.

Check the following V$SESSION_WAIT parameter columns:

- P1: File ID
- P2: Block ID
- P3: Class ID

Causes

To determine the possible causes, first query V$SESSION to identify the value of ROW_WAIT_OBJ# when the session waits for buffer busy waits. For example:

```
SELECT row_wait_obj#
  FROM V$SESSION
 WHERE EVENT = 'buffer busy waits';
```

To identify the object and object type contended for, query DBA_OBJECTS using the value for ROW_WAIT_OBJ# that is returned from V$SESSION. For example:

```
SELECT owner, object_name, subobject_name, object_type
  FROM DBA_OBJECTS
 WHERE data_object_id = &row_wait_obj;
```

Actions

The action required depends on the class of block contended for and the actual segment.

Segment Header

If the contention is on the segment header, then this is most likely free list contention.

Automatic segment-space management in locally managed tablespaces eliminates the need to specify the `PCTUSED`, `FREELISTS`, and `FREELIST GROUPS` parameters. If possible, switch from manual space management to automatic segment-space management (ASSM).

The following information is relevant if you are unable to use ASSM (for example, because the tablespace uses dictionary space management).

A free list is a list of free data blocks that usually includes blocks existing in several different extents within the segment. Free lists are composed of blocks in which free space has not yet reached PCTFREE or used space has shrunk below PCTUSED. Specify the number of process free lists with the `FREELISTS` parameter. The default value of `FREELISTS` is one. The maximum value depends on the data block size.

To find the current setting for free lists for that segment, run the following:

```
SELECT SEGMENT_NAME, FREELISTS
  FROM DBA_SEGMENTS
 WHERE SEGMENT_NAME = segment name
   AND SEGMENT_TYPE = segment type;
```

Set free lists, or increase the number of free lists. If adding more free lists does not alleviate the problem, then use free list groups (even in single instance this can make a difference). If using Oracle RAC, then ensure that each instance has its own free list group(s).

> **See Also:**
>
> *Oracle Database Concepts* for information about automatic segment-space management, free lists, `PCTFREE`, and `PCTUSED`

Data Block

If the contention is on tables or indexes (not the segment header):

- Check for right-hand indexes. These are indexes that are inserted into at the same point by many processes. For example, those that use sequence number generators for the key values.
- Consider using ASSM, global hash partitioned indexes, or increasing free lists to avoid multiple processes attempting to insert into the same block.

Undo Header

For contention on rollback segment header:

- If you are not using automatic undo management, then add more rollback segments.

Undo Block

For contention on rollback segment block:

- If you are not using automatic undo management, then consider making rollback segment sizes larger.

db file scattered read

This event signifies that the user process is reading buffers into the SGA buffer cache and is waiting for a physical I/O call to return. A `db file scattered read` issues a scattered read to read the data into multiple discontinuous memory locations. A scattered read is usually a multiblock read. It can occur for a fast full scan (of an index) in addition to a full table scan.

The `db file scattered read` wait event identifies that a full scan is occurring. When performing a full scan into the buffer cache, the blocks read are read into memory locations that are not physically adjacent to each other. Such reads are called scattered read calls, because the blocks are scattered throughout memory. This is why the corresponding wait event is called 'db file scattered read'. multiblock (up to `DB_FILE_MULTIBLOCK_READ_COUNT` blocks) reads due to full scans into the buffer cache show up as waits for 'db file scattered read'.

Check the following `V$SESSION_WAIT` parameter columns:

- `P1`: The absolute file number
- `P2`: The block being read
- `P3`: The number of blocks (should be greater than 1)

Actions

On a healthy system, physical read waits should be the biggest waits after the idle waits. However, also consider whether there are direct read waits (signifying full table scans with parallel query) or `db file scattered read` waits on an operational (OLTP) system that should be doing small indexed accesses.

Other things that could indicate excessive I/O load on the system include the following:

- Poor buffer cache hit ratio
- These wait events accruing most of the wait time for a user experiencing poor response time

Managing Excessive I/O

There are several ways to handle excessive I/O waits. In the order of effectiveness, these are as follows:

- Reduce the I/O activity by SQL tuning.
- Reduce the need to do I/O by managing the workload.
- Gather system statistics with `DBMS_STATS` package, allowing the query optimizer to accurately cost possible access paths that use full scans.
- Use Automatic Storage Management.
- Add more disks to reduce the number of I/Os for each disk.
- Alleviate I/O hot spots by redistributing I/O across existing disks.

The first course of action should be to find opportunities to reduce I/O. Examine the SQL statements being run by sessions waiting for these events and statements causing high

physical I/Os from V$SQLAREA. Factors that can adversely affect the execution plans causing excessive I/O include the following:

- Improperly optimized SQL
- Missing indexes
- High degree of parallelism for the table (skewing the optimizer toward scans)
- Lack of accurate statistics for the optimizer
- Setting the value for DB_FILE_MULTIBLOCK_READ_COUNT initialization parameter too high which favors full scans

Inadequate I/O Distribution

Besides reducing I/O, also examine the I/O distribution of files across the disks. Is I/O distributed uniformly across the disks, or are there hot spots on some disks? Are the number of disks sufficient to meet the I/O needs of the database?

See the total I/O operations (reads and writes) by the database, and compare those with the number of disks used. Remember to include the I/O activity of LGWR and ARCH processes.

Finding the SQL Statement executed by Sessions Waiting for I/O

Use the following query to determine, at a point in time, which sessions are waiting for I/O:

```
SELECT SQL_ADDRESS, SQL_HASH_VALUE
  FROM V$SESSION
 WHERE EVENT LIKE 'db file%read';
```

Finding the Object Requiring I/O

To determine the possible causes, first query V$SESSION to identify the value of ROW_WAIT_OBJ# when the session waits for db file scattered read. For example:

```
SELECT row_wait_obj#
  FROM V$SESSION
 WHERE EVENT = 'db file scattered read';
```

To identify the object and object type contended for, query DBA_OBJECTS using the value for ROW_WAIT_OBJ# that is returned from V$SESSION. For example:

```
SELECT owner, object_name, subobject_name, object_type
  FROM DBA_OBJECTS
 WHERE data_object_id = &row_wait_obj;
```

db file sequential read

This event signifies that the user process is reading a buffer into the SGA buffer cache and is waiting for a physical I/O call to return. A sequential read is a single-block read.

Single block I/Os are usually the result of using indexes. Rarely, full table scan calls could get truncated to a single block call because of extent boundaries, or buffers present in the buffer cache. These waits would also show up as db file sequential read.

Check the following V$SESSION_WAIT parameter columns:

- `P1`: The absolute file number
- `P2`: The block being read
- `P3`: The number of blocks (should be 1)

> **See Also:**
>
> "db file scattered read" for information about managing excessive I/O, inadequate I/O distribution, and finding the SQL causing the I/O and the segment the I/O is performed on.

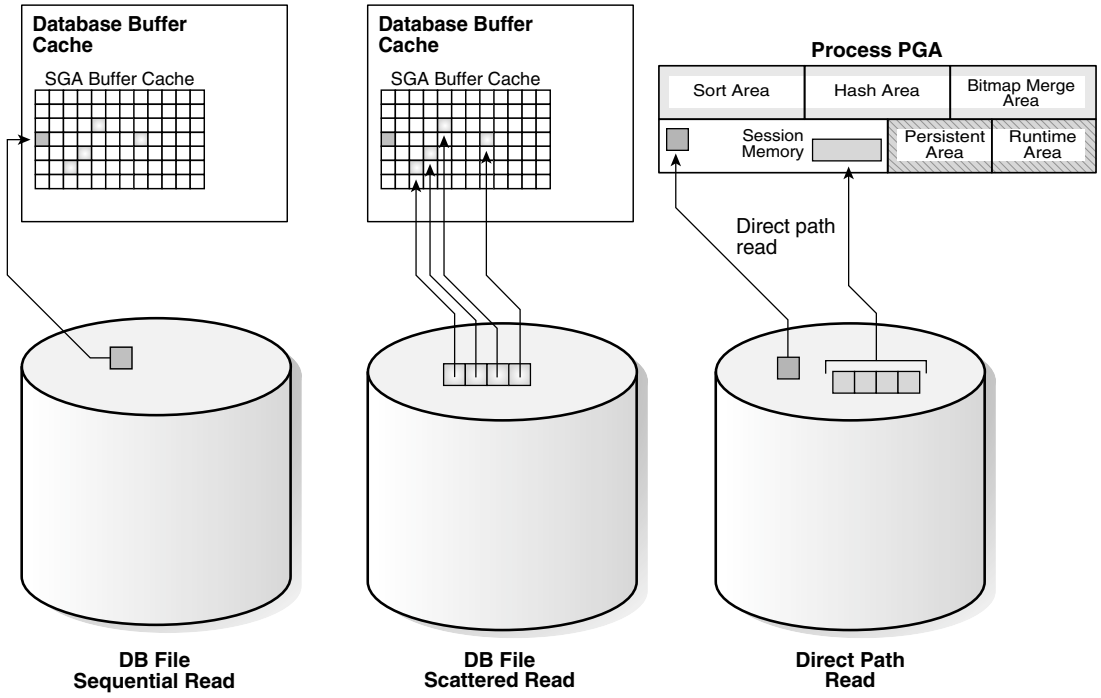
Actions

On a healthy system, physical read waits should be the biggest waits after the idle waits. However, also consider whether there are `db file sequential reads` on a large data warehouse that should be seeing mostly full table scans with parallel query.

The following figure shows differences between these wait events:

- `db file sequential read` (single block read into one SGA buffer)
- `db file scattered read` (multiblock read into many discontinuous SGA buffers)
- `direct read` (single or multiblock read into the PGA, bypassing the SGA)

Figure 10-1 Scattered Read, Sequential Read, and Direct Path Read



direct path read and direct path read temp

When a session is reading buffers from disk directly into the PGA (opposed to the buffer cache in SGA), it waits on this event. If the I/O subsystem does not support asynchronous I/Os, then each wait corresponds to a physical read request.

If the I/O subsystem supports asynchronous I/O, then the process is able to overlap issuing read requests with processing the blocks existing in the PGA. When the process attempts to access a block in the PGA that has not yet been read from disk, it then issues a wait call and updates the statistics for this event. Hence, the number of waits is not necessarily the same as the number of read requests (unlike `db file scattered read` and `db file sequential read`).

Check the following `V$SESSION_WAIT` parameter columns:

- `P1`: File_id for the read call
- `P2`: Start block_id for the read call
- `P3`: Number of blocks in the read call

Causes

This situation occurs in the following situations:

- The sorts are too large to fit in memory and some of the sort data is written out directly to disk. This data is later read back in, using direct reads.
- Parallel execution servers are used for scanning data.
- The server process is processing buffers faster than the I/O system can return the buffers. This can indicate an overloaded I/O system.

Actions

The `file_id` shows if the reads are for an object in `TEMP` tablespace (sorts to disk) or full table scans by parallel execution servers. This wait is the largest wait for large data warehouse sites. However, if the workload is not a Decision Support Systems (DSS) workload, then examine why this situation is happening.

Sorts to Disk

Examine the SQL statement currently being run by the session experiencing waits to see what is causing the sorts. Query `V$TEMPSEG_USAGE` to find the SQL statement that is generating the sort. Also query the statistics from `V$SESSTAT` for the session to determine the size of the sort. See if it is possible to reduce the sorting by tuning the SQL statement. If `WORKAREA_SIZE_POLICY` is `MANUAL`, then consider increasing the `SORT_AREA_SIZE` for the system (if the sorts are not too big) or for individual processes. If `WORKAREA_SIZE_POLICY` is `AUTO`, then investigate whether to increase `PGA_AGGREGATE_TARGET`.

Full Table Scans

If tables are defined with a high degree of parallelism, then this setting could skew the optimizer to use full table scans with parallel execution servers. Check the object being read into using the direct path reads. If the full table scans are a valid part of the workload, then ensure that the I/O subsystem is adequate for the degree of

parallelism. Consider using disk striping if you are not already using it or Oracle Automatic Storage Management (Oracle ASM).

Hash Area Size

For query plans that call for a hash join, excessive I/O could result from having `HASH_AREA_SIZE` too small. If `WORKAREA_SIZE_POLICY` is `MANUAL`, then consider increasing the `HASH_AREA_SIZE` for the system or for individual processes. If `WORKAREA_SIZE_POLICY` is `AUTO`, then investigate whether to increase `PGA_AGGREGATE_TARGET`.

> **See Also:**
>
> - "Managing Excessive I/O" in the section "db file scattered read"

direct path write and direct path write temp

When a process is writing buffers directly from PGA (as opposed to the DBWR writing them from the buffer cache), the process waits on this event for the write call to complete. Operations that could perform direct path writes include sorts on disk, parallel DML operations, direct-path `INSERT`s, parallel create table as select, and some LOB operations.

Like direct path reads, the number of waits is not the same as number of write calls issued if the I/O subsystem supports asynchronous writes. The session waits if it has processed all buffers in the PGA and cannot continue work until an I/O request completes.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about direct-path inserts

Check the following `V$SESSION_WAIT` parameter columns:

- `P1`: File_id for the write call
- `P2`: Start block_id for the write call
- `P3`: Number of blocks in the write call

Causes

This happens in the following situations:

- Sorts are too large to fit in memory and are written to disk
- Parallel DML are issued to create/populate objects
- Direct path loads

Actions

For large sorts see "Sorts To Disk".

For parallel DML, check the I/O distribution across disks and ensure that the I/O subsystem is adequately configured for the degree of parallelism.

enqueue (enq:) waits

Enqueues are locks that coordinate access to database resources. This event indicates that the session is waiting for a lock that is held by another session.

The name of the enqueue is included as part of the wait event name, in the form `enq: enqueue_type - related_details`. In some cases, the same enqueue type can be held for different purposes, such as the following related `TX` types:

- `enq: TX - allocate ITL entry`
- `enq: TX - contention`
- `enq: TX - index contention`
- `enq: TX - row lock contention`

The `V$EVENT_NAME` view provides a complete list of all the `enq:` wait events.

You can check the following `V$SESSION_WAIT` parameter columns for additional information:

- `P1`: Lock `TYPE` (or name) and `MODE`
- `P2`: Resource identifier ID1 for the lock
- `P3`: Resource identifier ID2 for the lock

> **See Also:**
>
> *Oracle Database Reference* for more information about Oracle Database enqueues

Finding Locks and Lock Holders

Query `V$LOCK` to find the sessions holding the lock. For every session waiting for the event enqueue, there is a row in `V$LOCK` with `REQUEST` <> `0`. Use one of the following two queries to find the sessions holding the locks and waiting for the locks.

If there are enqueue waits, you can see these using the following statement:

```
SELECT * FROM V$LOCK WHERE request > 0;
```

To show only holders and waiters for locks being waited on, use the following:

```
SELECT DECODE(request,0,'Holder: ','Waiter: ') ||
          sid sess, id1, id2, lmode, request, type
   FROM V$LOCK
 WHERE (id1, id2, type) IN (SELECT id1, id2, type FROM V$LOCK WHERE request > 0)
   ORDER BY id1, request;
```

Actions

The appropriate action depends on the type of enqueue.

If the contended-for enqueue is the ST enqueue, then the problem is most likely to be dynamic space allocation. Oracle Database dynamically allocates an extent to a segment when there is no more free space available in the segment. This enqueue is only used for dictionary managed tablespaces.

To solve contention on this resource:

- Check to see whether the temporary (that is, sort) tablespace uses `TEMPFILES`. If not, then switch to using `TEMPFILES`.
- Switch to using locally managed tablespaces if the tablespace that contains segments that are growing dynamically is dictionary managed.
- If it is not possible to switch to locally managed tablespaces, then ST enqueue resource usage can be decreased by changing the next extent sizes of the growing objects to be large enough to avoid constant space allocation. To determine which segments are growing constantly, monitor the `EXTENTS` column of the `DBA_SEGMENTS` view for all `SEGMENT_NAMEs`.
- Preallocate space in the segment, for example, by allocating extents using the `ALTER TABLE ALLOCATE EXTENT` SQL statement.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for detailed information on `TEMPFILES` and locally managed tablespaces
> - *Oracle Database Administrator's Guide* for more information about getting space usage details

The HW enqueue is used to serialize the allocation of space beyond the high water mark of a segment.

- `V$SESSION_WAIT.P2` / `V$LOCK.ID1` is the tablespace number.
- `V$SESSION_WAIT.P3` / `V$LOCK.ID2` is the relative data block address (dba) of segment header of the object for which space is being allocated.

If this is a point of contention for an object, then manual allocation of extents solves the problem.

The most common reason for waits on TM locks tend to involve foreign key constraints where the constrained columns are not indexed. Index the foreign key columns to avoid this problem.

These are acquired exclusive when a transaction initiates its first change and held until the transaction does a `COMMIT` or `ROLLBACK`.

- Waits for TX in mode 6: occurs when a session is waiting for a row level lock that is held by another session. This occurs when one user is updating or deleting a row, which another session wants to update or delete. This type of TX enqueue wait corresponds to the wait event `enq: TX - row lock contention`.

  The solution is to have the first session holding the lock perform a `COMMIT` or `ROLLBACK`.
- Waits for TX in mode 4 can occur if the session is waiting for an ITL (interested transaction list) slot in a block. This happens when the session wants to lock a row in the

block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block. Usually, Oracle Database dynamically adds another ITL slot. This may not be possible if there is insufficient free space in the block to add an ITL. If so, the session waits for a slot with a TX enqueue in mode 4. This type of TX enqueue wait corresponds to the wait event `enq: TX - allocate ITL entry`.

The solution is to increase the number of ITLs available, either by changing the `INITRANS` or `MAXTRANS` for the table (either by using an `ALTER` statement, or by re-creating the table with the higher values).

- Waits for TX in mode 4 can also occur if a session is waiting due to potential duplicates in `UNIQUE` index. If two sessions try to insert the same key value the second session has to wait to see if an `ORA-0001` should be raised or not. This type of TX enqueue wait corresponds to the wait event `enq: TX - row lock contention`.

  The solution is to have the first session holding the lock perform a `COMMIT` or `ROLLBACK`.

- Waits for TX in mode 4 can also occur if the session is waiting due to shared bitmap index fragment. Bitmap indexes index key values and a range of rowids. Each entry in a bitmap index can cover many rows in the actual table. If two sessions want to update rows covered by the same bitmap index fragment, then the second session waits for the first transaction to either `COMMIT` or `ROLLBACK` by waiting for the TX lock in mode 4. This type of TX enqueue wait corresponds to the wait event `enq: TX - row lock contention`.
- Waits for TX in Mode 4 can also occur waiting for a `PREPARED` transaction.
- Waits for TX in mode 4 also occur when a transaction inserting a row in an index has to wait for the end of an index block split being done by another transaction. This type of TX enqueue wait corresponds to the wait event `enq: TX - index contention`.

> **See Also:**
>
> *Oracle Database Development Guide* for more information about referential integrity and locking data explicitly

events in wait class other

This event belong to Other wait class and typically should not occur on a system. This event is an aggregate of all other events in the Other wait class, such as `latch free`, and is used in the `V$SESSION_EVENT` and `V$SERVICE_EVENT` views only. In these views, the events in the Other wait class will not be maintained individually in every session. Instead, these events will be rolled up into this single event to reduce the memory used for maintaining statistics on events in the Other wait class.

free buffer waits

This wait event indicates that a server process was unable to find a free buffer and has posted the database writer to make free buffers by writing out dirty buffers. A dirty buffer is a buffer whose contents have been modified. Dirty buffers are freed for reuse when DBWR has written the blocks to disk.

Causes

DBWR may not be keeping up with writing dirty buffers in the following situations:

- The I/O system is slow.
- There are resources it is waiting for, such as latches.
- The buffer cache is so small that DBWR spends most of its time cleaning out buffers for server processes.
- The buffer cache is so big that one DBWR process is not enough to free enough buffers in the cache to satisfy requests.

Actions

If this event occurs frequently, then examine the session waits for DBWR to see whether there is anything delaying DBWR.

If it is waiting for writes, then determine what is delaying the writes and fix it. Check the following:

- Examine `V$FILESTAT` to see where most of the writes are happening.
- Examine the host operating system statistics for the I/O system. Are the write times acceptable?

If I/O is slow:

- Consider using faster I/O alternatives to speed up write times.
- Spread the I/O activity across large number of spindles (disks) and controllers.

It is possible DBWR is very active because the cache is too small. Investigate whether this is a probable cause by looking to see if the buffer cache hit ratio is low. Also use the `V$DB_CACHE_ADVICE` view to determine whether a larger cache size would be advantageous.

If the cache size is adequate and the I/O is evenly spread, then you can potentially modify the behavior of DBWR by using asynchronous I/O or by using multiple database writers.

Consider Multiple Database Writer (DBWR) Processes or I/O Slaves

Configuring multiple database writer processes, or using I/O slaves, is useful when the transaction rates are high or when the buffer cache size is so large that a single DBW*n* process cannot keep up with the load.

The `DB_WRITER_PROCESSES` initialization parameter lets you configure multiple database writer processes (from DBW0 to DBW9 and from DBWa to DBWj). Configuring multiple DBWR processes distributes the work required to identify buffers to be written, and it also distributes the I/O load over these processes. Multiple db writer processes are highly recommended for systems with multiple CPUs (at least one db writer for every 8 CPUs) or multiple processor groups (at least as many db writers as processor groups).

Based upon the number of CPUs and the number of processor groups, Oracle Database either selects an appropriate default setting for `DB_WRITER_PROCESSES` or adjusts a user-specified setting.

If it is not practical to use multiple DBWR processes, then Oracle Database provides a facility whereby the I/O load can be distributed over multiple slave processes. The DBWR process is the only process that scans the buffer cache LRU list for blocks to be written out. However,

the I/O for those blocks is performed by the I/O slaves. The number of I/O slaves is determined by the parameter `DBWR_IO_SLAVES`.

`DBWR_IO_SLAVES` is intended for scenarios where you cannot use multiple `DB_WRITER_PROCESSES` (for example, where you have a single CPU). I/O slaves are also useful when asynchronous I/O is not available, because the multiple I/O slaves simulate nonblocking, asynchronous requests by freeing DBWR to continue identifying blocks in the cache to be written. Asynchronous I/O at the operating system level, if you have it, is generally preferred.

DBWR I/O slaves are allocated immediately following database open when the first I/O request is made. The DBWR continues to perform all of the DBWR-related work, apart from performing I/O. I/O slaves simply perform the I/O on behalf of DBWR. The writing of the batch is parallelized between the I/O slaves.

> **Note:**
>
> Implementing `DBWR_IO_SLAVES` requires that extra shared memory be allocated for I/O buffers and request queues. Multiple DBWR processes cannot be used with I/O slaves. Configuring I/O slaves forces only one DBWR process to start.

Configuring multiple DBWR processes benefits performance when a single DBWR process cannot keep up with the required workload. However, before configuring multiple DBWR processes, check whether asynchronous I/O is available and configured on the system. If the system supports asynchronous I/O but it is not currently used, then enable asynchronous I/O to see if this alleviates the problem. If the system does not support asynchronous I/O, or if asynchronous I/O is configured and there is still a DBWR bottleneck, then configure multiple DBWR processes.

> **Note:**
>
> If asynchronous I/O is not available on your platform, then asynchronous I/O can be disabled by setting the `DISK_ASYNCH_IO` initialization parameter to `FALSE`.

Using multiple DBWRs parallelizes the gathering and writing of buffers. Therefore, multiple DBW*n* processes should deliver more throughput than one DBWR process with the same number of I/O slaves. For this reason, the use of I/O slaves has been deprecated in favor of multiple DBWR processes. I/O slaves should only be used if multiple DBWR processes cannot be configured.

Idle Wait Events

These events belong to Idle wait class and indicate that the server process is waiting because it has no work. This usually implies that if there is a bottleneck, then the bottleneck is not for database resources. The majority of the idle events should be ignored when tuning, because they do not indicate the nature of the performance bottleneck. Some idle events can be useful in indicating what the bottleneck is not. An example of this type of event is the most commonly encountered idle wait-event `SQL`

`Net message from client`. This and other idle events (and their categories) are listed in Table 10-2.

Table 10-2 Idle Wait Events

Wait Name	Background Process Idle Event	User Process Idle Event	Parallel Query Idle Event	Shared Server Idle Event	Oracle Real Application Clusters Idle Event
`dispatcher timer`	.	.	.	X	.
`pipe get`	.	X	.	.	.
`pmon timer`	X	.	.	.	.
`PX Idle Wait`	.	.	X	.	.
`PX Deq Credit: need buffer`	.	.	X	.	.
`rdbms ipc message`	X	.	.	.	.
`shared server idle wait`	.	.	.	X	.
`smon timer`	X	.	.	.	.
`SQL*Net message from client`	.	X	.	.	.

> **See Also:**
>
> *Oracle Database Reference* for explanations of each idle wait event

latch events

A latch is a low-level internal lock used by Oracle Database to protect memory structures. The latch free event is updated when a server process attempts to get a latch, and the latch is unavailable on the first attempt.

There is a dedicated latch-related wait event for the more popular latches that often generate significant contention. For those events, the name of the latch appears in the name of the wait event, such as `latch: library cache` or `latch: cache buffers chains`. This enables you to quickly figure out if a particular type of latch is responsible for most of the latch-related contention. Waits for all other latches are grouped in the generic `latch free` wait event.

> **See Also:**
>
> *Oracle Database Concepts* for more information on latches and internal locks

Actions

This event should only be a concern if latch waits are a significant portion of the wait time on the system as a whole, or for individual users experiencing problems.

- Examine the resource usage for related resources. For example, if the library cache latch is heavily contended for, then examine the hard and soft parse rates.
- Examine the SQL statements for the sessions experiencing latch contention to see if there is any commonality.

Check the following `V$SESSION_WAIT` parameter columns:

- `P1`: Address of the latch
- `P2`: Latch number
- `P3`: Number of times process has slept, waiting for the latch

Example: Find Latches Currently Waited For

```
SELECT EVENT, SUM(P3) SLEEPS, SUM(SECONDS_IN_WAIT) SECONDS_IN_WAIT
  FROM V$SESSION_WAIT
 WHERE EVENT LIKE 'latch%'
  GROUP BY EVENT;
```

A problem with the previous query is that it tells more about session tuning or instant instance tuning than instance or long-duration instance tuning.

The following query provides more information about long duration instance tuning, showing whether the latch waits are significant in the overall database time.

```
SELECT EVENT, TIME_WAITED_MICRO,
       ROUND(TIME_WAITED_MICRO*100/S.DBTIME,1) PCT_DB_TIME
  FROM V$SYSTEM_EVENT,
   (SELECT VALUE DBTIME FROM V$SYS_TIME_MODEL WHERE STAT_NAME = 'DB time') S
 WHERE EVENT LIKE 'latch%'
 ORDER BY PCT_DB_TIME ASC;
```

A more general query that is not specific to latch waits is the following:

```
SELECT EVENT, WAIT_CLASS,
      TIME_WAITED_MICRO,ROUND(TIME_WAITED_MICRO*100/S.DBTIME,1) PCT_DB_TIME
  FROM V$SYSTEM_EVENT E, V$EVENT_NAME N,
    (SELECT VALUE DBTIME FROM V$SYS_TIME_MODEL WHERE STAT_NAME = 'DB time') S
   WHERE E.EVENT_ID = N.EVENT_ID
    AND N.WAIT_CLASS NOT IN ('Idle', 'System I/O')
  ORDER BY PCT_DB_TIME ASC;
```

Table 10-3 Latch Wait Event

Latch	SGA Area	Possible Causes	Look For:
Shared pool, library cache	Shared pool	Lack of statement reuse Statements not using bind variables Insufficient size of application cursor cache Cursors closed explicitly after each execution Frequent logins and logoffs Underlying object structure being modified (for example truncate) Shared pool too small	Sessions (in `V$SESSTAT`) with high: • `parse time CPU` • `parse time elapsed` • Ratio of `parse count (hard)` / `execute count` • Ratio of `parse count (total)` / `execute count` Cursors (in `V$SQLAREA`/`V$SQLSTATS`) with: • High ratio of `PARSE_CALLS` / `EXECUTIONS` • `EXECUTIONS` = 1 differing only in literals in the `WHERE` clause (that is, no bind variables used) • High `RELOADS` • High `INVALIDATIONS` • Large (> 1mb) `SHARABLE_MEM`
cache buffers lru chain	Buffer cache LRU lists	Excessive buffer cache throughput. For example, inefficient SQL that accesses incorrect indexes iteratively (large index range scans) or many full table scans DBWR not keeping up with the dirty workload; hence, foreground process spends longer holding the latch looking for a free buffer Cache may be too small	Statements with very high logical I/O or physical I/O, using unselective indexes
cache buffers chains	Buffer cache buffers	Repeated access to a block (or small number of blocks), known as a hot block	Sequence number generation code that updates a row in a table to generate the number, rather than using a sequence number generator Index leaf chasing from very many processes scanning the same unselective index with very similar predicate Identify the segment the hot block belongs to
row cache objects			

Shared Pool and Library Cache Latch Contention

A main cause of shared pool or library cache latch contention is parsing. There are several techniques that you can use to identify unnecessary parsing and several types of unnecessary parsing:

This method identifies similar SQL statements that could be shared if literals were replaced with bind variables. The idea is to either:

- Manually inspect SQL statements that have only one execution to see whether they are similar:

```
SELECT SQL_TEXT
  FROM V$SQLSTATS
 WHERE EXECUTIONS < 4
 ORDER BY SQL_TEXT;
```

- Or, automate this process by grouping what may be similar statements. Estimate the number of bytes of a SQL statement that are likely the same, and group the SQL statements by this number of bytes. For example, the following example groups statements that differ only after the first 60 bytes.

```
SELECT SUBSTR(SQL_TEXT, 1, 60), COUNT(*)
  FROM V$SQLSTATS
 WHERE EXECUTIONS < 4
 GROUP BY SUBSTR(SQL_TEXT, 1, 60)
 HAVING COUNT(*) > 1;
```

- Or report distinct SQL statements that have the same execution plan. The following query selects distinct SQL statements that share the same execution plan at least four times. These SQL statements are likely to be using literals instead of bind variables.

```
SELECT SQL_TEXT FROM V$SQLSTATS WHERE PLAN_HASH_VALUE IN
  (SELECT PLAN_HASH_VALUE
     FROM V$SQLSTATS
    GROUP BY PLAN_HASH_VALUE HAVING COUNT(*) > 4)
  ORDER BY PLAN_HASH_VALUE;
```

Check the `V$SQLSTATS` view. Enter the following query:

```
SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS
  FROM V$SQLSTATS
ORDER BY PARSE_CALLS;
```

When the `PARSE_CALLS` value is close to the `EXECUTIONS` value for a given statement, you might be continually reparsing that statement. Tune the statements with the higher numbers of parse calls.

Identify unnecessary parse calls by identifying the session in which they occur. It might be that particular batch programs or certain types of applications do most of the reparsing. To achieve this goal, run the following query:

```
SELECT pa.SID, pa.VALUE "Hard Parses", ex.VALUE "Execute Count"
  FROM V$SESSTAT pa, V$SESSTAT ex
 WHERE pa.SID = ex.SID
   AND pa.STATISTIC#=(SELECT STATISTIC#
       FROM V$STATNAME WHERE NAME = 'parse count (hard)')
   AND ex.STATISTIC#=(SELECT STATISTIC#
       FROM V$STATNAME WHERE NAME = 'execute count')
   AND pa.VALUE > 0;
```

The result is a list of all sessions and the amount of reparsing they do. For each session identifier (SID), go to `V$SESSION` to find the name of the program that causes the reparsing.

> **Note:**
>
> Because this query counts all parse calls since instance startup, it is best to look for sessions with high *rates* of parse. For example, a connection which has been up for 50 days might show a high parse figure, but a second connection might have been up for 10 minutes and be parsing at a much faster rate.

The output is similar to the following:

```
   SID  Hard Parses  Execute Count
------  -----------  -------------
     7            1             20
     8            3          12690
     6           26            325
    11           84           1619
```

The `cache buffers lru chain` latches protect the lists of buffers in the cache. When adding, moving, or removing a buffer from a list, a latch must be obtained.

For symmetric multiprocessor (SMP) systems, Oracle Database automatically sets the number of LRU latches to a value equal to one half the number of CPUs on the system. For non-SMP systems, one LRU latch is sufficient.

Contention for the LRU latch can impede performance on SMP computers with a large number of CPUs. LRU latch contention is detected by querying `V$LATCH`, `V$SESSION_EVENT`, and `V$SYSTEM_EVENT`. To avoid contention, consider tuning the application, bypassing the buffer cache for DSS jobs, or redesigning the application.

The `cache buffers chains` latches are used to protect a buffer list in the buffer cache. These latches are used when searching for, adding, or removing a buffer from the buffer cache. Contention on this latch usually means that there is a block that is greatly contended for (known as a hot block).

To identify the heavily accessed buffer chain, and hence the contended for block, look at latch statistics for the `cache buffers chains` latches using the view `V$LATCH_CHILDREN`. If there is a specific `cache buffers chains` child latch that has many more `GETS`, `MISSES`, and `SLEEPS` when compared with the other child latches, then this is the contended for child latch.

This latch has a memory address, identified by the `ADDR` column. Use the value in the `ADDR` column joined with the `X$BH` table to identify the blocks protected by this latch. For example, given the address (`V$LATCH_CHILDREN.ADDR`) of a heavily contended latch, this queries the file and block numbers:

```
SELECT OBJ data_object_id, FILE#, DBABLK,CLASS, STATE, TCH
  FROM X$BH
 WHERE HLADDR = 'address of latch'
  ORDER BY TCH;
```

`X$BH.TCH` is a touch count for the buffer. A high value for `X$BH.TCH` indicates a hot block.

Many blocks are protected by each latch. One of these buffers will probably be the hot block. Any block with a high `TCH` value is a potential hot block. Perform this query several times, and identify the block that consistently appears in the output. After you have identified the hot block, query `DBA_EXTENTS` using the file number and block number, to identify the segment.

After you have identified the hot block, you can identify the segment it belongs to with the following query:

```
SELECT OBJECT_NAME, SUBOBJECT_NAME
  FROM DBA_OBJECTS
 WHERE DATA_OBJECT_ID = &obj;
```

In the query, `&obj` is the value of the `OBJ` column in the previous query on `X$BH`.

The `row cache objects` latches protect the data dictionary.

log file parallel write

This event involves writing redo records to the redo log files from the log buffer.

library cache pin

This event manages library cache concurrency. Pinning an object causes the heaps to be loaded into memory. If a client wants to modify or examine the object, the client must acquire a pin after the lock.

library cache lock

This event controls the concurrency between clients of the library cache. It acquires a lock on the object handle so that either:

- One client can prevent other clients from accessing the same object
- The client can maintain a dependency for a long time which does not allow another client to change the object

This lock is also obtained to locate an object in the library cache.

log buffer space

This event occurs when server processes are waiting for free space in the log buffer, because all the redo is generated faster than LGWR can write it out.

Actions

Modify the redo log buffer size. If the size of the log buffer is reasonable, then ensure that the disks on which the online redo logs reside do not suffer from I/O contention. The `log buffer space` wait event could be indicative of either disk I/O contention on the disks where the redo logs reside, or of a too-small log buffer. Check the I/O profile of the disks containing the redo logs to investigate whether the I/O system is the bottleneck. If the I/O system is not a problem, then the redo log buffer could be too small. Increase the size of the redo log buffer until this event is no longer significant.

log file switch

There are two wait events commonly encountered:

- log file switch (archiving needed)
- log file switch (checkpoint incomplete)

In both of the events, the LGWR cannot switch into the next online redo log file. All the commit requests wait for this event.

Actions

For the `log file switch` (`archiving needed`) event, examine why the archiver cannot archive the logs in a timely fashion. It could be due to the following:

- Archive destination is running out of free space.
- Archiver is not able to read redo logs fast enough (contention with the LGWR).
- Archiver is not able to write fast enough (contention on the archive destination, or not enough ARCH processes). If you have ruled out other possibilities (such as slow disks or a full archive destination) consider increasing the number of ARCn processes. The default is 2.
- If you have mandatory remote shipped archive logs, check whether this process is slowing down because of network delays or the write is not completing because of errors.

Depending on the nature of bottleneck, you might need to redistribute I/O or add more space to the archive destination to alleviate the problem. For the `log file switch` (`checkpoint incomplete`) event:

- Check if DBWR is slow, possibly due to an overloaded or slow I/O system. Check the DBWR write times, check the I/O system, and distribute I/O if necessary.
- Check if there are too few, or too small redo logs. If you have a few redo logs or small redo logs (for example, 2 x 100k logs), and your system produces enough redo to cycle through all of the logs before DBWR has been able to complete the checkpoint, then increase the size or number of redo logs.

log file sync

When a user session commits (or rolls back), the session's redo information must be flushed to the redo logfile by LGWR. The server process performing the `COMMIT` or `ROLLBACK` waits under this event for the write to the redo log to complete.

Actions

If this event's waits constitute a significant wait on the system or a significant amount of time waited by a user experiencing response time issues or on a system, then examine the average time waited.

If the average time waited is low, but the number of waits are high, then the application might be committing after every `INSERT`, rather than batching `COMMIT`s. Applications can reduce the wait by committing after 50 rows, rather than every row.

If the average time waited is high, then examine the session waits for the log writer and see what it is spending most of its time doing and waiting for. If the waits are because of slow I/O, then try the following:

- Reduce other I/O activity on the disks containing the redo logs, or use dedicated disks.
- Alternate redo logs on different disks to minimize the effect of the archiver on the log writer.
- Move the redo logs to faster disks or a faster I/O subsystem (for example, switch from RAID 5 to RAID 1).

- Consider using raw devices (or simulated raw devices provided by disk vendors) to speed up the writes.
- Depending on the type of application, it might be possible to batch `COMMIT`s by committing every *N* rows, rather than every row, so that fewer log file syncs are needed.

rdbms ipc reply

This event is used to wait for a reply from one of the background processes.

SQL*Net Events

The following events signify that the database process is waiting for acknowledgment from a database link or a client process:

- `SQL*Net break/reset to client`
- `SQL*Net break/reset to dblink`
- `SQL*Net message from client`
- `SQL*Net message from dblink`
- `SQL*Net message to client`
- `SQL*Net message to dblink`
- `SQL*Net more data from client`
- `SQL*Net more data from dblink`
- `SQL*Net more data to client`
- `SQL*Net more data to dblink`

If these waits constitute a significant portion of the wait time on the system or for a user experiencing response time issues, then the network or the middle-tier could be a bottleneck.

Events that are client-related should be diagnosed as described for the event `SQL*Net message from client`. Events that are dblink-related should be diagnosed as described for the event `SQL*Net message from dblink`.

SQL*Net message from client

Although this is an idle event, it is important to explain when this event can be used to diagnose what is not the problem. This event indicates that a server process is waiting for work from the client process. However, there are several situations where this event could accrue most of the wait time for a user experiencing poor response time. The cause could be either a network bottleneck or a resource bottleneck on the client process.

A network bottleneck can occur if the application causes a lot of traffic between server and client and the network latency (time for a round-trip) is high. Symptoms include the following:

- Large number of waits for this event
- Both the database and client process are idle (waiting for network traffic) most of the time

To alleviate network bottlenecks, try the following:

- Tune the application to reduce round trips.
- Explore options to reduce latency (for example, terrestrial lines opposed to `VSAT` links).
- Change system configuration to move higher traffic components to lower latency links.

If the client process is using most of the resources, then there is nothing that can be done in the database. Symptoms include the following:

- Number of waits might not be large, but the time waited might be significant
- Client process has a high resource usage

In some cases, you can see the wait time for a waiting user tracking closely with the amount of CPU used by the client process. The term client here refers to any process other than the database process (middle-tier, desktop client) in the n-tier architecture.

SQL*Net message from dblink

This event signifies that the session has sent a message to the remote node and is waiting for a response from the database link. This time could go up because of the following:

- Network bottleneck

  For information, see "SQL*Net message from client".

- Time taken to execute the SQL on the remote node

  It is useful to see the SQL being run on the remote node. Login to the remote database, find the session created by the database link, and examine the SQL statement being run by it.

- Number of round trip messages

  Each message between the session and the remote node adds latency time and processing overhead. To reduce the number of messages exchanged, use array fetches and array inserts.

SQL*Net more data to client

The server process is sending more data or messages to the client. The previous operation to the client was also a send.

> **See Also:**
>
> *Oracle Database Net Services Administrator's Guide* for a detailed discussion on network optimization

Tuning Instance Recovery Performance: Fast-Start Fault Recovery

This section describes instance recovery, and how Oracle's Fast-Start Fault Recovery improves availability in the event of a crash or instance failure. It also offers guidelines for tuning the time required to perform crash and instance recovery.

This section contains the following topics:

About Instance Recovery

Instance and crash recovery are the automatic application of redo log records to Oracle data blocks after a crash or system failure. During normal operation, if an instance is shut down cleanly (as when using a `SHUTDOWN IMMEDIATE` statement), rather than terminated abnormally, then the in-memory changes that have not been written to the data files on disk are written to disk as part of the checkpoint performed during shutdown.

However, if a single instance database crashes or if all instances of an Oracle RAC configuration crash, then Oracle Database performs crash recovery at the next startup. If one or more instances of an Oracle RAC configuration crash, then a surviving instance performs instance recovery automatically. Instance and crash recovery occur in two steps: cache recovery followed by transaction recovery.

The database can be opened as soon as cache recovery completes, so improving the performance of cache recovery is important for increasing availability.

Cache Recovery (Rolling Forward)

During the cache recovery step, Oracle Database applies all committed and uncommitted changes in the redo log files to the affected data blocks. The work required for cache recovery processing is proportional to the rate of change to the database (update transactions each second) and the time between checkpoints.

Transaction Recovery (Rolling Back)

To make the database consistent, the changes that were not committed at the time of the crash must be undone (in other words, rolled back). During the transaction recovery step, Oracle Database applies the rollback segments to undo the uncommitted changes.

Checkpoints and Cache Recovery

Periodically, Oracle Database records a checkpoint. A **checkpoint** is the highest system change number (SCN) such that all data blocks less than or equal to that SCN are known to be written out to the data files. If a failure occurs, then only the redo records containing changes at SCNs higher than the checkpoint need to be applied during recovery. The duration of cache recovery processing is determined by two factors: the number of data blocks that have changes at SCNs higher than the SCN of the checkpoint, and the number of log blocks that need to be read to find those changes.

How Checkpoints Affect Performance

Frequent checkpointing writes dirty buffers to the data files more often than otherwise, and so reduces cache recovery time in the event of an instance failure. If checkpointing is frequent, then applying the redo records in the redo log between the

current checkpoint position and the end of the log involves processing relatively few data blocks. This means that the cache recovery phase of recovery is fairly short.

However, in a high-update system, frequent checkpointing can reduce run-time performance, because checkpointing causes DBW*n* processes to perform writes.

Fast Cache Recovery Tradeoffs

To minimize the duration of cache recovery, you must force Oracle Database to checkpoint often, thus keeping the number of redo log records to be applied during recovery to a minimum. However, in a high-update system, frequent checkpointing increases the overhead for normal database operations.

If daily operational efficiency is more important than minimizing recovery time, then decrease the frequency of writes to data files due to checkpoints. This should improve operational efficiency, but also increase cache recovery time.

Configuring the Duration of Cache Recovery: FAST_START_MTTR_TARGET

The Fast-Start Fault Recovery feature reduces the time required for cache recovery, and makes the recovery bounded and predictable by limiting the number of dirty buffers and the number of redo records generated between the most recent redo record and the last checkpoint.

The foundation of Fast-Start Fault Recovery is the Fast-Start checkpointing architecture. Instead of conventional event-driven (that is, log switching) checkpointing, which does bulk writes, fast-start checkpointing occurs incrementally. Each DBW*n* process periodically writes buffers to disk to advance the checkpoint position. The oldest modified blocks are written first to ensure that every write lets the checkpoint advance. Fast-Start checkpointing eliminates bulk writes and the resultant I/O spikes that occur with conventional checkpointing.

With the Fast-Start Fault Recovery feature, the `FAST_START_MTTR_TARGET` initialization parameter simplifies the configuration of recovery time from instance or system failure. `FAST_START_MTTR_TARGET` specifies a target for the expected mean time to recover (MTTR), that is, the time (in seconds) that it should take to start up the instance and perform cache recovery. After `FAST_START_MTTR_TARGET` is set, the database manages incremental checkpoint writes in an attempt to meet that target. If you have chosen a practical value for `FAST_START_MTTR_TARGET`, you can expect your database to recover, on average, in approximately the number of seconds you have chosen.

> **Note:**
>
> You must disable or remove the `FAST_START_IO_TARGET`, `LOG_CHECKPOINT_INTERVAL`, and `LOG_CHECKPOINT_TIMEOUT` initialization parameters when using `FAST_START_MTTR_TARGET`. Setting these parameters interferes with the mechanisms used to manage cache recovery time to meet `FAST_START_MTTR_TARGET`.

Practical Values for FAST_START_MTTR_TARGET

The maximum value for FAST_START_MTTR_TARGET is 3600 seconds (one hour). If you set the value to more than 3600, then Oracle Database rounds it to 3600.

The following example shows how to set the value of FAST_START_MTTR_TARGET:

```
SQL> ALTER SYSTEM SET FAST_START_MTTR_TARGET=30;
```

In principle, the minimum value for FAST_START_MTTR_TARGET is one second. However, the fact that you can set FAST_START_MTTR_TARGET this low does not mean that this target can be achieved. There are practical limits to the minimum achievable MTTR target, due to such factors as database startup time.

The MTTR target that your database can achieve given the current value of FAST_START_MTTR_TARGET is called the **effective MTTR target**. You can view your current effective MTTR by viewing the TARGET_MTTR column of the V$INSTANCE_RECOVERY view.

The practical range of MTTR target values for your database is defined to be the range between the lowest achievable effective MTTR target for your database and the longest that startup and cache recovery will take in the worst-case scenario (that is, when the whole buffer cache is dirty). "Determine the Practical Range for FAST_START_MTTR_TARGET" describes the procedure for determining the range of achievable MTTR target values, one step in the process of tuning your FAST_START_MTTR_TARGET value.

> **Note:**
>
> It is usually not useful to set your FAST_START_MTTR_TARGET to a value outside the practical range. If your FAST_START_MTTR_TARGET value is shorter than the lower limit of the practical range, the effect is as if you set it to the lower limit of the practical range. In such a case, the effective MTTR target will be the best MTTR target the system can achieve, but checkpointing will be at a maximum, which can affect normal database performance. If you set FAST_START_MTTR_TARGET to a time longer than the practical range, the MTTR target will be no better than the worst-case situation.

Reducing Checkpoint Frequency to Optimize Run-Time Performance

To reduce the checkpoint frequency and optimize run-time performance, you can do the following:

- Set the value of FAST_START_MTTR_TARGET to 3600. This enables Fast-Start checkpointing and the Fast-Start Fault Recovery feature, but minimizes its effect on run-time performance while avoiding the need for performance tuning of FAST_START_MTTR_TARGET.
- Size your online redo log files according to the amount of redo your system generates. Try to switch logs at most every twenty minutes. Having your log files too small can increase checkpoint activity and reduce performance. Also note that all redo log files should be the same size.

> **See Also:**
>
> *Oracle Database Concepts* for detailed information about checkpoints

Monitoring Cache Recovery with V$INSTANCE_RECOVERY

The `V$INSTANCE_RECOVERY` view displays the current recovery parameter settings. You can also use statistics from this view to determine which factor has the greatest influence on checkpointing.

The following table lists those columns most useful in monitoring predicted cache recovery performance:

Table 10-4 V$INSTANCE_RECOVERY Columns

Column	Description
`TARGET_MTTR`	Effective MTTR target in seconds. This field is 0 if `FAST_START_MTTR_TARGET` is not specified.
`ESTIMATED_MTTR`	The current estimated MTTR in seconds, based on the current number of dirty buffers and log blocks. This field is always calculated, whether `FAST_START_MTTR_TARGET` is specified.

As part of the ongoing monitoring of your database, you can periodically compare `V$INSTANCE_RECOVERY.TARGET_MTTR` to your `FAST_START_MTTR_TARGET`. The two values should generally be the same if the `FAST_START_MTTR_TARGET` value is in the practical range. If `TARGET_MTTR` is consistently longer than `FAST_START_MTTR_TARGET`, then set `FAST_START_MTTR_TARGET` to a value no less than `TARGET_MTTR`. If `TARGET_MTTR` is consistently shorter, then set `FAST_START_MTTR_TARGET` to a value no greater than `TARGET_MTTR`.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `V$INSTANCE_RECOVERY` view

Tuning FAST_START_MTTR_TARGET and Using MTTR Advisor

To determine the appropriate value for `FAST_START_MTTR_TARGET` for your database, use the following four step process:

- Calibrate the FAST_START_MTTR_TARGET
- Determine the Practical Range for FAST_START_MTTR_TARGET
- Evaluate Different Target Values with MTTR Advisor
- Determine the Optimal Size for Redo Logs

Calibrate the FAST_START_MTTR_TARGET

The FAST_START_MTTR_TARGET initialization parameter causes the database to calculate internal system trigger values, in order to limit the length of the redo log and the number of dirty data buffers in the data cache. This calculation uses estimated time to read a redo block, estimates of the time to read and write a data block and characteristics of typical workload of the system, such as how many dirty buffers corresponds to how many change vectors, and so on.

Initially, internal defaults are used in the calculation. These defaults are replaced over time by data gathered on I/O performance during system operation and actual cache recoveries.

You will have to perform several instance recoveries in order to calibrate your FAST_START_MTTR_TARGET value properly. Before starting calibration, you must decide whether FAST_START_MTTR_TARGET is being calibrated for a database crash or a hardware crash. This is a consideration if your database files are stored in a file system or if your I/O subsystem has a memory cache, because there is a considerable difference in the read and write time to disk depending on whether the files are cached. The appropriate value for FAST_START_MTTR_TARGET will depend upon which type of crash is more important to recover from quickly.

To effectively calibrate FAST_START_MTTR_TARGET, ensure that you run the typical workload of the system for long enough, and perform several instance recoveries to ensure that the time to read a redo block and the time to read or write a data block during recovery are recorded accurately.

Determine the Practical Range for FAST_START_MTTR_TARGET

After calibration, you can perform tests to determine the practical range for FAST_START_MTTR_TARGET for your database.

Determining Lower Bound for FAST_START_MTTR_TARGET: Scenario

To determine the lower bound of the practical range, set FAST_START_MTTR_TARGET to 1, and start up your database. Then check the value of V$INSTANCE_RECOVERY.TARGET_MTTR, and use this value as a good lower bound for FAST_START_MTTR_TARGET. Database startup time, rather than cache recovery time, is usually the dominant factor in determining this limit.

For example, set the FAST_START_MTTR_TARGET to 1:

```
SQL> ALTER SYSTEM SET FAST_START_MTTR_TARGET=1;
```

Then, execute the following query immediately after opening the database:

```
SQL> SELECT TARGET_MTTR, ESTIMATED_MTTR
    FROM V$INSTANCE_RECOVERY;
```

Oracle Database responds with the following:

```
TARGET_MTTR ESTIMATED_MTTR
18          15
```

The TARGET_MTTR value of 18 seconds is the minimum MTTR target that the system can achieve, that is, the lowest practical value for FAST_START_MTTR_TARGET. This minimum is calculated based on the average database startup time.

The ESTIMATED_MTTR field contains the estimated mean time to recovery based on the current state of the running database. Because the database has just opened, the system contains few dirty buffers, so not much cache recovery would be required if the instance failed at this moment. That is why ESTIMATED_MTTR can, for the moment, be lower than the minimum possible TARGET_MTTR.

ESTIMATED_MTTR can be affected in the short term by recent database activity. Assume that you query V$INSTANCE_RECOVERY immediately after a period of heavy update activity in the database. Oracle Database responds with the following:

```
TARGET_MTTR ESTIMATED_MTTR
18          30
```

Now the effective MTTR target is still 18 seconds, and the estimated MTTR (if a crash happened at that moment) is 30 seconds. This is an acceptable result. This means that some checkpoints writes might not have finished yet, so the buffer cache contains more dirty buffers than targeted.

Now wait for sixty seconds and reissue the query to V$INSTANCE_RECOVERY. Oracle Database responds with the following:

```
TARGET_MTTR ESTIMATED_MTTR
18          25
```

The estimated MTTR at this time has dropped to 25 seconds, because some of the dirty buffers have been written out during this period

Determining Upper Bound for FAST_START_MTTR_TARGET

To determine the upper bound of the practical range, set FAST_START_MTTR_TARGET to 3600, and operate your database under a typical workload for a while. Then check the value of V$INSTANCE_RECOVERY.TARGET_MTTR. This value is a good upper bound for FAST_START_MTTR_TARGET.

The procedure is substantially similar to that in "Determining Lower Bound for FAST_START_MTTR_TARGET: Scenario".

Selecting Preliminary Value for FAST_START_MTTR_TARGET

After you have determined the practical bounds for the FAST_START_MTTR_TARGET parameter, select a preliminary value for the parameter. Choose a higher value within the practical range if your concern is with database performance, and a lower value within the practical range if your priority is shorter recovery times. The narrower the practical range, of course, the easier the choice becomes.

For example, if you discovered that the practical range was between 17 and 19 seconds, it would be quite simple to choose 19, because it makes relatively little difference in recovery time and at the same time minimizes the effect of checkpointing on system performance. However, if you found that the practical range was between 18 and 40 seconds, you might choose a compromise value of 30, and set the parameter accordingly:

```
SQL> ALTER SYSTEM SET FAST_START_MTTR_TARGET=30;
```

You might then go on to use the MTTR Advisor to determine an optimal value.

Evaluate Different Target Values with MTTR Advisor

After you have selected a preliminary value for FAST_START_MTTR_TARGET, you can use MTTR Advisor to evaluate the effect of different FAST_START_MTTR_TARGET settings on system performance, compared to your chosen setting.

Enabling MTTR Advisor

To enable MTTR Advisor, set the two initialization parameters STATISTICS_LEVEL and FAST_START_MTTR_TARGET.

STATISTICS_LEVEL governs whether all advisors are enabled and is not specific to MTTR Advisor. Ensure that it is set to TYPICAL or ALL. Then, when FAST_START_MTTR_TARGET is set to a nonzero value, the MTTR Advisor is enabled.

Using MTTR Advisor

After enabling MTTR Advisor, run a typical database workload for a while. When MTTR Advisor is ON, the database simulates checkpoint queue behavior under the current value of FAST_START_MTTR_TARGET, and up to four other different MTTR settings within the range of valid FAST_START_MTTR_TARGET values. (The database will in this case determine the valid range for FAST_START_MTTR_TARGET itself before testing different values in the range.)

Viewing MTTR Advisor Results: V$MTTR_TARGET_ADVICE

The dynamic performance view V$MTTR_TARGET_ADVICE lets you view statistics or advisories collected by MTTR Advisor.

The database populates V$MTTR_TARGET_ADVICE with advice about the effects of each of the FAST_START_MTTR_TARGET settings for your database. For each possible value of FAST_START_MTTR_TARGET, the row contains details about how many cache writes would be performed under the workload tested for that value of FAST_START_MTTR_TARGET.

Specifically, each row contains information about cache writes, total physical writes (including direct writes), and total I/O (including reads) for that value of FAST_START_MTTR_TARGET, expressed both as a total number of operations and a ratio compared to the operations under your chosen FAST_START_MTTR_TARGET value. For instance, a ratio of 1.2 indicates 20% more cache writes.

Knowing the effect of different FAST_START_MTTR_TARGET settings on cache write activity and other I/O enables you to decide better which FAST_START_MTTR_TARGET value best fits your recovery and performance needs.

If MTTR Advisor is currently on, then V$MTTR_TARGET_ADVICE shows the Advisor information collected. If MTTR Advisor is currently OFF, then the view shows information collected the last time MTTR Advisor was ON since database startup, if any. If the database has been restarted since the last time the MTTR Advisor was used, or if it has never been used, the view will not show any rows.

> **See Also:**
>
> *Oracle Database Reference* for the column details of the `V$MTTR_TARGET_ADVICE` view

Determine the Optimal Size for Redo Logs

You can use the `V$INSTANCE_RECOVERY` view column `OPTIMAL_LOGFILE_SIZE` to determine the size of your online redo logs. This field shows the redo log file size in megabytes that is considered optimal based on the current setting of `FAST_START_MTTR_TARGET`. If this field consistently shows a value greater than the size of your smallest online log, then you should configure all your online logs to be at least this size.

Note, however, that the redo log file size affects the MTTR. In some cases, you may be able to refine your choice of the optimal `FAST_START_MTTR_TARGET` value by re-running the MTTR Advisor with your suggested optimal log file size.

Part III
Tuning Database Memory

This part contains the following chapters:

11

Database Memory Allocation

This chapter describes memory allocation in Oracle Database and the various methods for managing memory.

This chapter contains the following topics:

About Database Memory Caches and Other Memory Structures

Oracle Database stores information in memory caches and on disk. Memory access is much faster than disk access. Disk access (physical I/O) takes a significant amount of time, compared to memory access, typically in the order of 10 milliseconds. Physical I/O also increases the CPU resources required due to the path length in device drivers and operating system event schedulers. For this reason, it is more efficient for data requests of frequently accessed objects to be perform by memory, rather than also requiring disk access. Proper sizing and effective use of Oracle Database memory caches greatly improves database performance.

The main Oracle Database memory caches that affect performance include:

- Database buffer cache

  The database buffer cache stores data blocks read from disk.

- Redo log buffer

  The redo log buffer stores redo entries of changes made to data blocks in the buffer cache.

- Shared pool

  The shared pool caches many different types of data and is mainly comprised of the following components:

  - Library cache
  - Data dictionary cache
  - Server result cache

- Large pool

  The large pool provides large memory allocations for the following Oracle Database features:

  - Shared server architecture
  - Parallel query
  - Recovery Manager (RMAN)

- Java pool

  The Java pool stores session-specific Java code and Java Virtual Machine (JVM) data.

- Streams pool

  The Streams pool provides memory for Oracle Advanced Queuing (AQ) and replication processes.

- Process-private memory

  Process-private memory includes memory used for operations such as sorting and hash joins.

- In-Memory Column Store (IM column store)

  Starting in Oracle Database 12c Release 1 (12.1.0.2), the IM column store is an optional, static SGA pool that stores copies of tables and partitions. In the IM column store, data is stored in a special columnar format, which improves performance of operations such as scans, joins, and aggregations.

> **Note:**
>
> The IM column store does not replace the buffer cache, but acts as a supplement so that both memory areas can store the same data in different formats.

> **See Also:**
>
> *Oracle Database Concepts* for information about the Oracle Database memory architecture

Database Memory Management Methods

The goal of memory management is to reduce the physical I/O overhead as much as possible, either by making it more likely that the required data is in memory, or by making the process of retrieving the required data more efficient. To achieve this goal, proper sizing and effective use of Oracle Database memory caches is essential.

Oracle Database provides the following methods to manage database memory:

- Automatic Memory Management
- Automatic Shared Memory Management
- Manual Shared Memory Management
- Automatic PGA Memory Management
- Manual PGA Memory Management

Automatic Memory Management

Automatic memory management enables Oracle Database to manage and tune the database memory automatically. In automatic memory management mode, management of the shared global area (SGA) and program global area (instance PGA) memory is handled completely by Oracle Database. This method is the most automated and is strongly recommended by Oracle. Before setting any memory pool sizes manually, strongly consider using automatic memory management.

For information about using automatic memory management, see "Using Automatic Memory Management".

Automatic Shared Memory Management

If automatic memory management is disabled, then Oracle Database uses automatic shared memory management to manage SGA memory. In this mode, Oracle Database automatically distributes memory to individual SGA components based on a target size that you set for the total SGA memory.

For information about using automatic shared memory management, see "Using Automatic Shared Memory Management".

Manual Shared Memory Management

If both automatic memory management and automatic shared memory management are disabled, then you must manage SGA memory manually by sizing the individual memory pools in the SGA. Although this mode enables you to exercise complete control over how SGA memory is distributed, it requires the most effort because the SGA components must be manually tuned on an ongoing basis.

For information about using manual shared memory management, see "Sizing the SGA Components Manually".

Automatic PGA Memory Management

If automatic memory management is disabled, then Oracle Database uses automatic PGA memory management to manage PGA memory. In this mode, Oracle Database automatically distributes memory to work areas in the instance PGA based on a target size that you set for the total PGA memory.

For information about automatic PGA memory management, see Tuning the Program Global Area .

Manual PGA Memory Management

If both automatic memory management and automatic PGA memory management are disabled, then you must manage PGA memory manually by adjusting the portion of PGA memory dedicated to each work area. This method can be very difficult because the workload is always changing and is not recommended by Oracle. Although manual PGA memory management is supported by Oracle Database, Oracle strongly recommends using automatic memory management or automatic PGA memory management instead.

Using Automatic Memory Management

To use automatic memory management, set the following initialization parameters:

- `MEMORY_TARGET`

  The `MEMORY_TARGET` initialization parameter specifies the target memory size. The database tunes to the value specified for this parameter, redistributing memory as needed between the SGA and the instance PGA. This parameter is dynamic, so its value can be changed at any time without restarting the database.

- `MEMORY_MAX_TARGET`

  The `MEMORY_MAX_TARGET` initialization parameter specifies the maximum memory size. The value specified for this parameter serves as the limit to which the `MEMORY_TARGET` initialization parameter can be set. This parameter is static, so its value cannot be changed after instance startup.

If you need tuning advice for the `MEMORY_TARGET` parameter, then use the `V$MEMORY_TARGET_ADVICE` view.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about using automatic memory management

Monitoring Memory Management

Table 11-1 lists the views that provide information about memory resize operations.

Table 11-1 Memory Management Views

View	Description
`V$MEMORY_CURRENT_RESIZE_OPS`	Displays information about memory resize operations (both automatic and manual) that are currently in progress.
`V$MEMORY_DYNAMIC_COMPONENTS`	Displays information about the current sizes of all dynamically-tuned memory components, including the total sizes of the SGA and instance PGA.
`V$MEMORY_RESIZE_OPS`	Displays information about the last 800 completed memory resize operations (both automatic and manual). This does not include operations that are currently in progress.
`V$MEMORY_TARGET_ADVICE`	Displays tuning advice for the `MEMORY_TARGET` initialization parameter.

See Also:

Oracle Database Reference for more information about these views

12

Tuning the System Global Area

This chapter describes how to tune the System Global Area (SGA). If you are using automatic memory management to manage the database memory on your system, then there is no need to tune the SGA as described in this chapter.

This chapter contains the following topics:

Using Automatic Shared Memory Management

Automatic shared memory management simplifies the configuration of the SGA by automatically distributing the memory in the SGA for the following memory pools:

- Database buffer cache (default pool)
- Shared pool
- Large pool
- Java pool
- Streams pool

Automatic shared memory management is controlled by the `SGA_TARGET` parameter. Changes in the value of the `SGA_TARGET` parameter automatically resize these memory pools. If these memory pools are set to nonzero values, then automatic shared memory management uses these values as minimum levels. Oracle recommends that you set the minimum values based on the minimum amount of memory an application component requires to function properly.

The following memory caches are manually-sized components and are not controlled by automatic shared memory management:

- Redo log buffer

  The redo log buffer is sized using the `LOG_BUFFER` initialization parameter, as described in "Configuring the Redo Log Buffer".

- Other buffer caches (such as `KEEP`, `RECYCLE`, and other nondefault block size)

  The `KEEP` pool is sized using the `DB_KEEP_CACHE_SIZE` initialization parameter, as described in "Configuring the KEEP Pool".

  The `RECYCLE` pool is sized using the `DB_RECYCLE_CACHE_SIZE` initialization parameter, as described in "Configuring the RECYCLE Pool".

- Fixed SGA and other internal allocations

Fixed SGA and other internal allocations are sized using the `DB_nK_CACHE_SIZE` initialization parameter.

The memory allocated to these memory caches is deducted from the value of the `SGA_TARGET` parameter when automatic shared memory management computes the values of the automatically-tuned memory pools.

The following sections describe how to access and set the value of the `SGA_TARGET` parameter:

- User Interfaces for Setting the SGA_TARGET Parameter
- Setting the SGA_TARGET Parameter

> **See Also:**
>
> - *Oracle Database Concepts* for information about the SGA
> - *Oracle Database Administrator's Guide* for information about managing the SGA
> - *Oracle Database Administrator's Guide* for information about using initialization parameters

User Interfaces for Setting the SGA_TARGET Parameter

This section describes the user interfaces for setting the value of the `SGA_TARGET` parameter.

This section contains the following topics:

- Setting the SGA_TARGET Parameter in Oracle Enterprise Manager Cloud Control
- Setting the SGA_TARGET Parameter in the Command-Line Interface

Setting the SGA_TARGET Parameter in Oracle Enterprise Manager Cloud Control

You can change the value of the `SGA_TARGET` parameter in Oracle Enterprise Manager Cloud Control (Cloud Control) by accessing the SGA Size Advisor from the Memory Parameters SGA page.

Setting the SGA_TARGET Parameter in the Command-Line Interface

You can change the value of the `SGA_TARGET` parameter in the command-line interface by querying the `V$SGA_TARGET_ADVICE` view and using the `ALTER SYSTEM` command.

Setting the SGA_TARGET Parameter

This section describes how to enable and disable automatic shared memory management by setting the value of the `SGA_TARGET` parameter.

This section contains the following topics:

Enabling Automatic Shared Memory Management

To enable automatic shared memory management, set the following initialization parameters:

- `STATISTICS_LEVEL` to `TYPICAL` or `ALL`
- `SGA_TARGET` to a nonzero value

  The `SGA_TARGET` parameter can be set to a value that is less than or equal to the value of the `SGA_MAX_SIZE` initialization parameter. Set the value of the `SGA_TARGET` parameter to the amount of memory that you intend to dedicate to the SGA.

Disabling Automatic Shared Memory Management

To disable automatic shared memory management, set the value of the `SGA_TARGET` parameter dynamically to 0 at instance startup.

This disables automatic shared memory management and the current auto-tuned sizes will be used for each memory pool. If necessary, you can manually resize each memory pool, as described in "Sizing the SGA Components Manually".

Unified Program Global Area

The unified program global area (PGA) pool is a shared global area (SGA) component that is used for PGA work areas by certain pluggable databases (PDBs) that have smaller SGA target values per-CPU compared to the multi-tenant container database (CDB).

The PGA allows the replacement of an Autonomous Transaction Processing (ATP) pluggable database (PDB) with an Autonomous Data Warehouse (ADW) PDB. This is achieved by transparently allocating some PGA for the ADW PDB from the SGA.

In an ATP-D environment, the memory is split between SGA and PGA usage. The SGA is supported by the large pages that are typically reserved during the virtual machine or host start-up time or in the early stage before the memory is fragmented. The SGA portion of system memory is configured based on transaction process requirements, typically 65% for SGA and 35% for PGA. However, ADW pluggable databases require more private memory and are typically in the range of 35% for SGA and 65% for PGA. Unfortunately, because the system is configured for ATP, the memory from huge pages (configured for SGA) cannot be released for PGA needs. It is difficult to reclaim the large pages in the future once it is fragmented.

The unified PGA pool is a new construct that allows the shared global area (SGA) to be used for pluggable databases' (PDB) program global area (PGA). The unified PGA only requests full granules from the buffer cache when it needs to grow, and it gives up full granules when no longer needed. At instance start-up, the PGA can have a minimum size of 0 or the size defined by an underscore size parameter.

> **Note:**
>
> Because only full granules can be consumed by the unified PGA pool, care is taken as to how often the granules are requested for growth and how soon they are given back to the system list.

Sizing the SGA Components Manually

If the system is not using automatic memory management or automatic shared memory management, then you must manually configure the sizes of the following SGA components:

- Database buffer cache

  The database buffer cache is sized using the `DB_CACHE_SIZE` initialization parameter, as described in "Configuring the Database Buffer Cache".

- Shared pool

  The shared pool is sized using the `SHARED_POOL_SIZE` initialization parameter, as described in "Configuring the Shared Pool".

- Large pool

  The large pool is sized using the `LARGE_POOL_SIZE` initialization parameter, as described in "Configuring the Large Pool".

- Java pool

  The Java pool is sized using the `JAVA_POOL_SIZE` initialization parameter.

- Streams pool

  The Streams pool is sized using the `STREAMS_POOL_SIZE` initialization parameter.

- IM column store

  The IM column store is sized using the `INMEMORY_SIZE` initialization parameter.

The values for these parameters are also dynamically configurable using the `ALTER SYSTEM` statement.

Before configuring the sizes of these SGA components, take the following considerations into account:

- SGA Sizing Unit
- Maximum Size of the SGA
- Application Considerations
- Operating System Memory Use
- Iteration During Configuration

> **See Also:**
>
> - *Oracle Database Java Developer's Guide* for information about Java memory usage and the `JAVA_POOL_SIZE` initialization parameter
> - *Oracle Database In-Memory Guide* for information about the `INMEMORY_SIZE` initialization parameter

SGA Sizing Unit

Memory for the buffer cache, shared pool, large pool, and Java pool is allocated in units of granules. If the SGA size is less than 1 GB, then the granule size is 4MB. If the SGA size is greater than 1 GB, the granule size changes to 16MB. The granule size is calculated and fixed when the database instance starts up. The size does not change during the lifetime of the instance.

To view the granule size that is currently being used for the SGA, use the `V$SGA_DYNAMIC_COMPONENTS` view. The same granule size is used for all dynamic components in the SGA.

Maximum Size of the SGA

The maximum amount of memory usable by the database instance is determined at instance startup by the value of the `SGA_MAX_SIZE` initialization parameter. You can expand the total SGA size to a value equal to the `SGA_MAX_SIZE` parameter. The value of the `SGA_MAX_SIZE` parameter defaults to the aggregate setting of all the SGA components.

If the value of the `SGA_MAX_SIZE` parameter is not set, then decrease the size of one cache and reallocate that memory to another cache if necessary. Alternatively, you can set the value of the `SGA_MAX_SIZE` parameter to be larger than the sum of all of the SGA components, such as the buffer cache and the shared pool. Doing so enables you to dynamically increase a cache size without having to decrease the size of another cache.

> **Note:**
>
> The value of the `SGA_MAX_SIZE` parameter cannot be dynamically resized.

Application Considerations

When configuring memory, size the memory caches appropriately based on the application's needs. Conversely, tuning the application's use of the memory caches can greatly reduce resource requirements. Efficient use of the memory caches also reduces the load on related resources, such as latches, CPU, and the I/O system.

For optimal performance, consider the following:

- Design the cache to use the operating system and database resources in the most efficient manner.

- Allocate memory to Oracle Database memory structures to best reflect the needs of the application.
- If changes or additions are made to an existing application, resize Oracle Database memory structures to meet the needs of the modified application.
- If the application uses Java, investigate whether the default configuration for the Java pool needs to be modified.

> **See Also:**
>
> *Oracle Database Java Developer's Guide* for information about Java memory usage

Operating System Memory Use

For most operating systems, it is important to consider the following when configuring memory:

- Reduce Paging
- Fit the SGA into Main Memory
- Allow Adequate Memory to Individual Users

> **See Also:**
>
> Your operating system hardware and software documentation, and the Oracle documentation specific to your operating system, for more information on tuning operating system memory usage

Reduce Paging

Paging occurs when an operating system transfers memory-resident pages to disk solely to load new pages into memory. Many operating systems page to accommodate large amounts of information that do not fit into real memory. On most operating systems, paging reduces performance.

To determine whether significant paging is occurring on the host system, use operating system utilities to examine the operating system. If significant paging is occurring, then the total system memory may not be large enough to hold the memory caches for which memory is allocated. Consider either increasing the total memory on the system, or decreasing the amount of memory allocated.

Fit the SGA into Main Memory

Because the purpose of the SGA is to store data in memory for fast access, the SGA should reside in the main memory. If pages of the SGA are swapped to disk, then the data is no longer quickly accessible. On most operating systems, the disadvantage of paging significantly outweighs the advantage of a large SGA.

This section contains the following topics:

Viewing SGA Memory Allocation

To view how much memory is allocated to the SGA and each of its internal structures, use the `SHOW SGA` statement in SQL*Plus as shown in the following example:

```
SQL> SHOW SGA
```

The output of this statement might look like the following:

```
Total System Global Area  840205000 bytes
Fixed Size                   279240 bytes
Variable Size             520093696 bytes
Database Buffers          318767104 bytes
Redo Buffers                1064960 bytes
```

Locking the SGA into Physical Memory

To prevent the SGA from being paged out, consider locking the SGA into physical memory by enabling the `LOCK_SGA` parameter. The database does not use the `MEMORY_TARGET` and `MEMORY_MAX_TARGET` parameters when the `LOCK_SGA` parameter is enabled.

Allow Adequate Memory to Individual Users

When sizing the SGA, ensure that you allow enough memory for the individual server processes and any other programs running on the system.

Iteration During Configuration

Configuring memory allocation involves distributing available memory to Oracle Database memory structures, depending on the needs of the application. The distribution of memory to Oracle Database structures can affect the amount of physical I/O necessary for Oracle Database to operate properly. Having a proper initial memory configuration provides an indication of whether the I/O system is effectively configured.

After the initial pass through the memory configuration process, it may be necessary to repeat the steps of memory allocation. Subsequent passes enable you to make adjustments to earlier steps, based on changes in subsequent steps. For example, decreasing the size of the buffer cache enables you to increase the size of another memory structure, such as the shared pool.

Monitoring Shared Memory Management

Table 12-1 lists the views that provide information about SGA resize operations.

Table 12-1 Shared Memory Management Views

View	Description
V$SGA_CURRENT_RESIZE_OPS	Displays information about SGA resize operations that are currently in progress.

Table 12-1 (Cont.) Shared Memory Management Views

View	Description
V$SGA_RESIZE_OPS	Displays information about the last 800 completed SGA resize operations. This does not include operations that are currently in progress.
V$SGA_DYNAMIC_COMPONENTS	Displays information about the dynamic components in the SGA. This view summarizes information of all completed SGA resize operations that occurred after instance startup.
V$SGA_DYNAMIC_FREE_MEMORY	Displays information about the amount of SGA memory available for future dynamic SGA resize operations.

See Also:

Oracle Database Reference for information about these views

Improving Query Performance with the In-Memory Column Store

The In-Memory Column Store (IM column store) is an optional portion of the system global area (SGA) that stores copies of tables, partitions, and other database objects in columnar format, and this columnar data is optimized for rapid scans. As the IM column store stores database objects in memory, Oracle Database can perform scans, queries, joins, and aggregates on that data much faster as compared to performing these operations on a data that is stored on a disk.

Note:

- The IM column store and database buffer cache store the same data, but in different formats. The IM column store does not replace the row-based storage in the database buffer cache, but supplements it for achieving better query performance.
- The IM column store is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

See Also:

Oracle Database In-Memory Guide for more information about the IM column store

Enabling High Performance Data Streaming with the Memoptimized Rowstore

The Memoptimized Rowstore enables high performance data streaming for applications, such as Internet of Things (IoT).

This section contains the following topics:

About the Memoptimized Rowstore

The Memoptimized Rowstore enables high performance data streaming for applications, such as Internet of Things (IoT) applications, which typically stream small amounts of data in single-row inserts from a large number of clients simultaneously and also query data for clients at a very high frequency.

The Memoptimized Rowstore provides the following functionality:

- Fast ingest

  Fast ingest optimizes the processing of high-frequency, single-row data inserts into a database. Fast ingest uses the large pool for buffering the inserts before writing them to disk, so as to improve data insert performance.

- Fast lookup

  Fast lookup enables fast retrieval of data from a database for high-frequency queries. Fast lookup uses a separate memory area in the SGA called the *memoptimize pool* for buffering the data queried from tables, so as to improve query performance.

> **Note:**
>
> For using fast lookup, you must allocate appropriate memory size to the memoptimize pool using the `MEMOPTIMIZE_POOL_SIZE` initialization parameter.

> **See Also:**
>
> - "Using Fast Ingest"
> - "Using Fast Lookup"

Using Fast Ingest

Fast ingest optimizes the processing of high-frequency, single-row data inserts into database from applications, such as Internet of Things (IoT) applications.

Fast ingest uses the `MEMOPTIMIZE_WRITE` hint to insert data into tables specified as `MEMOPTIMIZE FOR WRITE`. The database temporarily buffers these inserts in the large pool and automatically commits the changes at the time of writing these buffered inserts to disk. The changes cannot be rolled back.

The inserts using fast ingest are also known as *deferred inserts*, because they are initially buffered in the large pool and later written to disk asynchronously by background processes.

Steps for using fast ingest for inserting data into a table

The following are the steps for using fast ingest for inserting data into a table:

1. Enable a table for fast ingest by specifying the `MEMOPTIMIZE FOR WRITE` clause in the `CREATE TABLE` or `ALTER TABLE` statement.

   ```
   SQL> create table test_fast_ingest (
   id number primary key,
   test_col varchar2(15))
   segment creation immediate
   memoptimize for write;

   Table created.
   ```

   See "Enabling a Table for Fast Ingest" for more information.

2. Enable fast ingest for inserts by specifying the `MEMOPTIMIZE_WRITE` hint in the `INSERT` statement.

   The following is not how fast ingest is meant to be used, but demonstrates the mechanism.

   ```
   SQL> insert /*+ memoptimize_write */ into test_fast_ingest values
   (1, 'test');

   1 row created

   SQL> insert /*+ memotimize_write */ into test_fast_ingest values
   (2, 'test');

   1 row created
   ```

   See "Specifying a Hint for Using Fast Ingest for Data Inserts" for more information.

The result of the two inserts above is to write data to the ingest buffer in the large pool of the SGA. At some point, that data is flushed to the `TEST_FAST_INGEST` table. Until that happens, the data is not durable.

Because the purpose of fast-ingest is to support high performance data streaming, a more realistic architecture would involve having one or more application or ingest servers collecting data and batching inserts to the database.

The first time an insert is run, the fast ingest area is allocated from the large pool. The amount of memory allocated is written to the `alert.log`.

```
Memoptimize Write allocated 2055M from large pool
```

If the request fails to allocate even the minimal memory requirement, then an error message is written to the `alert.log`.

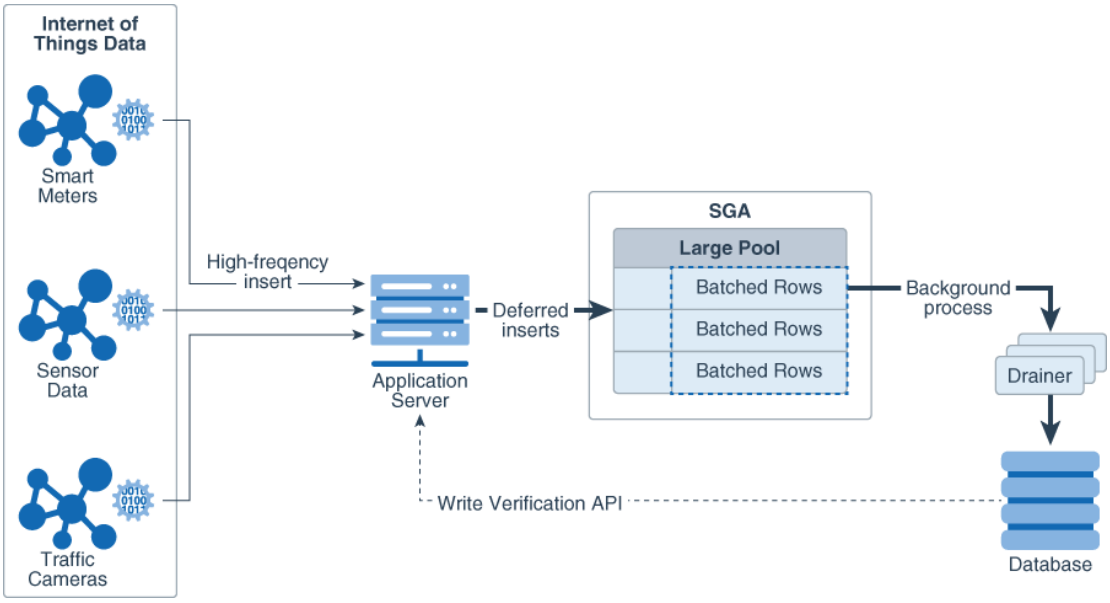
```
Memoptimize Write disabled. Unable to allocate sufficient memory from large
pool.
```

Details about Fast Ingest

The intent of fast-ingest is to support applications that generate lots of informational data that has important value in the aggregate but that doesn't necessarily require full ACID guarantees. Many applications in the Internet of Things (IoT) have a rapid *"fire and forget"* type workload, such as sensor data, smart meter data or even traffic cameras. For these applications, data might be collected and written to the database in high volumes for later analysis.

The following diagram shows how this might work with the Memoptimized Rowstore – Fast Ingest feature.

Figure 12-1 Fast-Ingest with high-frequency inserts.



The ingested data is batched in the large pool and is not immediately written to the database. Thus, the ingest process is very fast. Very large volumes of data can be ingested efficiently without having to process individual rows. However, if the database goes down before the ingested data is written out to the database files, it is possible to lose data.

Fast ingest is very different from normal Oracle Database transaction processing where data is logged and never lost once "written" to the database (i.e. committed). In order to achieve the maximum ingest throughput, the normal Oracle transaction mechanisms are bypassed, and it is the responsibility of the application to check to see that all data was indeed written to the database. Special APIs have been added that can be called to check if the data has been written to the database.

The `commit` operation has no meaning in the context of fast ingest, because it is not a transaction in the traditional Oracle sense. There is no ability to rollback the inserts. You also

cannot query the data until it has been flushed from the fast ingest buffers to disk. You can see some administrative information about the fast ingest buffers by querying the view `V$MEMOPTIMIZE_WRITE_AREA`.

You can also use the packages `DBMS_MEMOPTIMIZE` and `DBMS_MEMOPTIMIZE_ADMIN` to perform functions like flushing fast ingest data from the large pool and determining the sequence id of data that has been flushed.

Index operations and constraint checking is done only when the data is written from the fast ingest area in the large pool to disk. If primary key violations occur when the background processes write data to disk, then the database will not write those rows to the database.

Assuming (for most applications but not all) that all inserted data needs to be written to the database, it is critical that the application insert process checks to see that the inserted data has actually been written to the database before destroying that data. Only when that confirmation has occurred can the data be deleted from the inserting process.

See Also:

- Prerequisites for Fast Ingest Table
- Enabling a Table for Fast Ingest
- Specifying a Hint for Using Fast Ingest for Data Inserts
- Managing Fast Ingest Data in the Large Pool
- Disabling a Table for Fast Ingest
- *Oracle Database Concepts* for more information about the *deferred insert* mechanism

Prerequisites for Fast Ingest Table

Fast Ingest is not supported for tables with certain characteristics, objects, or partitioning.

When the Autonomous Database supports an item and does not have that limitation, it is so noted.

- Tables with the following characteristics cannot use fast ingest:
  - disk compression
  - in-memory compression
  - function-based indexes
  - domain indexes
  - bitmap indexes
  - bitmap join indexes
  - ref types
  - varray types

  - OID$ types
  - unused columns
  - LOBs
  - triggers
  - binary columns
  - foreign keys
  - row archival
  - invisible columns [Autonomous Database supports virtual columns.]
- The following objects cannot use fast ingest.
  - Temporary tables
  - Nested tables
  - Index organized tables
  - External tables
  - Materialized views with on-demand refresh
  - Sub-partitioning is not supported. [Autonomous Database supports sub-partitioning.]
- The following partitioning types are not supported.
  - REFERENCE
  - SYSTEM
  - INTERVAL [Autonomous Database supports this.]
  - AUTOLIST [Autonomous Database supports this.]

The following are some additional considerations for fast ingest:

- Assuming (for most applications but not all) that all inserted data needs to be written to the database, it is critical that the application implement process checks to see that the inserted data has actually been written to the database before destroying that data. Only when that confirmation has occurred can the data be deleted from the inserting process.
- Because fast ingest buffers data in the large pool, there is a possibility of data loss in the event of a system failure. To avoid data loss, a client must keep a local copy of the data after performing inserts, so that it can replay the inserts in the event of a system failure before the data is written to disk. A client can use the `DBMS_MEMOPTIMIZE` package subprograms to track the durability of the inserts. After inserts are written to disk, a client can destroy its local copy of the inserted data.
- Queries do not read data from the large pool, hence data inserted using fast ingest cannot be queried until it is written to disk.
- Index operations are supported by fast ingest similar to the regular inserts. However, for fast ingest, database performs index operations while writing data to disk, and not while writing data into the large pool.
- The size allocated to the fast ingest buffers in the Large pool is fixed once created. If the buffer fills, further ingest waits until the background processes drain the buffer.

> **Note:**
>
> A table can be configured for using both fast ingest and fast lookup.

Enabling a Table for Fast Ingest

You can enable a table for fast ingest by specifying the `MEMOPTIMIZE FOR WRITE` clause in the `CREATE TABLE` or `ALTER TABLE` statement.

To enable a table for fast ingest:

1. In SQL*Plus, log in to the database as a user with `ALTER TABLE` privileges.
2. Run the `CREATE TABLE` or `ALTER TABLE` statement with the `MEMOPTIMIZE FOR WRITE` clause.

   The following example creates a new table `test_fast_ingest` and enables it for fast ingest:

   ```
   CREATE TABLE test_fast_ingest (
        id        NUMBER(5) PRIMARY KEY,
        test_col  VARCHAR2(15))
     SEGMENT CREATION IMMEDIATE
     MEMOPTIMIZE FOR WRITE;
   ```

   The following example enables the existing table `hr.employees` for fast ingest:

   ```
   ALTER TABLE hr.employees MEMOPTIMIZE FOR WRITE;
   ```

Specifying a Hint for Using Fast Ingest for Data Inserts

You can use fast ingest for data inserts by specifying the `MEMOPTIMIZE_WRITE` hint in `INSERT` statements.

Prerequisites

This task assumes:

- The table is already enabled for fast ingest.
- The optimizer is allowed to use hints, meaning `optimizer_ignore_hints=FALSE`.

To use fast ingest for data inserts:

1. In SQL*Plus, log in to the database as a user with the privileges to insert data into tables.
2. Run the `INSERT` statement with the `MEMOPTIMIZE_WRITE` hint for a table that is already enabled for fast ingest.

For example:

```
INSERT  /*+ MEMOPTIMIZE_WRITE */ INTO test_fast_ingest VALUES (1,'test');
```

> **See Also:**
>
> "Enabling a Table for Fast Ingest"

Disabling a Table for Fast Ingest

You can disable a table for fast ingest by specifying the `NO MEMOPTIMIZE FOR WRITE` clause in the `ALTER TABLE` statement.

To disable a table for fast ingest:

1. In SQL*Plus, log in to the database as a user with the `ALTER TABLE` privileges.
2. Run the `ALTER TABLE` statement with the `NO MEMOPTIMIZE FOR WRITE` clause.

   The following example disables the table `hr.employees` for fast ingest:

   ```
   ALTER TABLE hr.employees NO MEMOPTIMIZE FOR WRITE;
   ```

Managing Fast Ingest Data in the Large Pool

You can view the fast ingest data in the large pool using the `V$MEMOPTIMIZE_WRITE_AREA` view. You can also view and control the fast ingest data in the large pool using the subprograms of the packages `DBMS_MEMOPTIMIZE` and `DBMS_MEMOPTIMIZE_ADMIN`.

Overview of the V$MEMOPTIMIZE_WRITE_AREA view

The `V$MEMOPTIMIZE_WRITE_AREA` view provides the following information about the memory usage and data inserts in the large pool by fast ingest:

- Total amount of memory allocated for fast ingest data in the large pool
- Total amount of memory currently used by fast ingest data in the large pool
- Total amount of memory currently free for storing fast ingest data in the large pool
- Number of fast ingest insert operations for which data is still in the large pool and is yet to be written to disk
- Number of clients currently using fast ingest for inserting data into the database

> **See Also:**
>
> *Oracle Database Reference* for information about the `V$MEMOPTIMIZE_WRITE_AREA` view

Overview of the DBMS_MEMOPTIMIZE package subprograms

You can use the following subprograms of the `DBMS_MEMOPTIMIZE` package to view and control the fast ingest data in the large pool:

Subprogram	Description
`GET_APPLY_HWM_SEQID`	Returns the low high-water mark (low HWM) of sequence numbers of data records that are successfully written to disk by all the sessions.
`GET_WRITE_HWM_SEQID`	Returns the high-water mark (HWM) sequence number of the data record that is written to the large pool for the current session.
`WRITE_END`	Flushes all the fast ingest data from the large pool to disk for the current session.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information about the `DBMS_MEMOPTIMIZE` package

Overview of the DBMS_MEMOPTIMIZE_ADMIN package subprograms

You can use the following subprograms of the `DBMS_MEMOPTIMIZE_ADMIN` package to control the fast ingest data in the large pool:

Subprogram	Description
`WRITES_FLUSH`	Flushes all the fast ingest data from the large pool to disk for all the sessions.

See Also:

Oracle Database PL/SQL Packages and Types Reference for information about the `DBMS_MEMOPTIMIZE_ADMIN` package

Using Fast Lookup

Fast lookup enables fast data retrieval from database tables for applications, such as Internet of Things (IoT) applications.

Fast lookup uses a hash index that is stored in the SGA buffer area called *memoptimize pool* to provide fast access to blocks of tables permanently pinned in the buffer cache, thus avoiding disk I/O and improving query performance.

Steps for using fast lookup for a table

The following are the steps for using fast lookup for a table:

1. Enable the memoptimize pool

This task is achieved by setting the `MEMOPTIMIZE_POOL_SIZE` initialization parameter to a non-zero value.

See "Enabling the Memoptimize Pool" for more information.

2. Enable a table for fast lookup

   This task is achieved by specifying the `MEMOPTIMIZE FOR READ` clause in the `CREATE TABLE` or `ALTER TABLE` statement.

   See "Enabling a Table for Fast Lookup" for more information.

Limitations for using fast lookup

The following are the limitations for using fast lookup:

- Tables enabled for fast lookup cannot be compressed.
- Tables enabled for fast lookup must have a primary key constraint enabled.

> **Note:**
>
> A table can be configured for using both fast ingest and fast lookup.

> **See Also:**
>
> - Enabling the Memoptimize Pool
> - Enabling a Table for Fast Lookup
> - Disabling a Table for Fast Lookup
> - Managing Fast Lookup Data in the Memoptimize Pool
> - *Oracle Database Concepts* for information about the memoptimize pool memory architecture
> - *Oracle Database Reference* for information about the `MEMOPTIMIZE_POOL_SIZE` initialization parameter

Enabling the Memoptimize Pool

You must enable the memoptimize pool before using fast lookup. The memoptimize pool resides in the SGA, and stores the data and hash index for the tables that are enabled for fast lookup.

Prerequisites

This task assumes that the `COMPATIBLE` initialization parameter is set to `18.0.0` or higher.

To enable the memoptimize pool:

1. In SQL*Plus, log in to the database as a user with administrative privileges.
2. Set the `MEMOPTIMIZE_POOL_SIZE` initialization parameter to a non-zero value. The minimum setting is `100 MB`. When you set this initialization parameter in a server

parameter file (SPFILE) using the `ALTER SYSTEM` statement, you must specify `SCOPE=SPFILE`.

For example, the following statement sets the memoptimize pool size to 10 GB:

```
ALTER SYSTEM SET MEMOPTIMIZE_POOL_SIZE = 10G SCOPE=SPFILE;
```

3. Restart the database for the change to take effect.

Example: Enabling the Memoptimize Pool

Assume that the `MEMOPTIMIZE_POOL_SIZE` initialization parameter is initially set to 0. The following example enables the memoptimize pool by setting the `MEMOPTIMIZE_POOL_SIZE` to 10 GB:

```
SQL> SHOW PARAMETER MEMOPTIMIZE_POOL_SIZE

NAME                   TYPE         VALUE
---------------------  -----------  -----
memoptimize_pool_size  big integer  0

SQL> ALTER SYSTEM SET MEMOPTIMIZE_POOL_SIZE=10G SCOPE=SPFILE;

System altered.

SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.

SQL> STARTUP
ORACLE instance started.

Total System Global Area 1.1832E+10 bytes
Fixed Size                  9010864 bytes
Variable Size            1.1799E+10 bytes
Database Buffers           16777216 bytes
Redo Buffers                7766016 bytes
Database mounted.
Database opened.

SQL> SHOW PARAMETER MEMOPTIMIZE_POOL_SIZE

NAME                   TYPE         VALUE
---------------------  -----------  -----
memoptimize_pool_size  big integer  10G
```

> **Note:**
>
> The `MEMOPTIMIZE_POOL_SIZE` value does count toward `SGA_TARGET`, but the database does not grow and shrink the memoptimize pool automatically. For example, if `SGA_TARGET` is 10 GB, and if `MEMOPTIMIZE_POOL_SIZE` is 1 GB, then a total of 9 GB is available for SGA memory other than the memoptimize pool.

> **See Also:**
>
> - *Oracle Database Concepts* for information about the memoptimize pool memory architecture
> - *Oracle Database Reference* for information about the `MEMOPTIMIZE_POOL_SIZE` initialization parameter

Enabling a Table for Fast Lookup

You can enable a table for fast lookup by specifying the `MEMOPTIMIZE FOR READ` clause in the `CREATE TABLE` or `ALTER TABLE` statement.

Prerequisites

This task assumes that the memoptimize pool is enabled.

To enable a table for fast lookup:

1. In SQL*Plus, log in to the database as a user with `ALTER TABLE` privileges.
2. Run the `CREATE TABLE` or `ALTER TABLE` statement with the `MEMOPTIMIZE FOR READ` clause for the table that needs to be enabled for fast lookup.

   The following example creates a new table `test_fast_lookup` and enables it for fast lookup:

   ```
   CREATE TABLE test_fast_lookup (
        id        NUMBER(5) PRIMARY KEY,
        test_col  VARCHAR2(15))
     MEMOPTIMIZE FOR READ;
   ```

   The following example enables the existing table `hr.employees` for fast lookup:

   ```
    ALTER TABLE hr.employees MEMOPTIMIZE FOR READ;
   ```

> **See Also:**
>
> - Enabling the Memoptimize Pool
> - Disabling a Table for Fast Lookup
> - Managing Fast Lookup Data in the Memoptimize Pool

Disabling a Table for Fast Lookup

You can disable a table for fast lookup by specifying the `NO MEMOPTIMIZE FOR READ` clause in the `ALTER TABLE` statement.

Prerequisites

This task assumes that a table is already enabled for fast lookup.

To disable a table for fast lookup:

1. In SQL*Plus, log in to the database as a user with the `ALTER TABLE` privileges.
2. Run the `ALTER TABLE` statement with the `NO MEMOPTIMIZE FOR READ` clause for the table that needs to be disabled for fast lookup.

   The following example disables the `hr.employees` table for fast lookup:

```
ALTER TABLE hr.employees NO MEMOPTIMIZE FOR READ;
```

> **See Also:**
>
> "Enabling a Table for Fast Lookup"

Managing Fast Lookup Data in the Memoptimize Pool

The memoptimize pool stores the data (*fast lookup data*) of all the tables that are enabled for fast lookup. You can explicitly delete or populate fast lookup data for a table in the memoptimize pool using the `DBMS_MEMOPTIMIZE` package subprograms.

Overview of the DBMS_MEMOPTIMIZE package subprograms

The following are the `DBMS_MEMOPTIMIZE` package subprograms that can be used to delete or populate fast lookup data for a table in the memoptimize pool:

Subprogram	Description
`DROP_OBJECT`	Removes fast lookup data for a table from the memoptimize pool.
`POPULATE`	Populates fast lookup data for a table in the memoptimize pool.

See Also:

- "Enabling a Table for Fast Lookup"
- "Enabling the Memoptimize Pool"
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_MEMOPTIMIZE` package
- *Oracle Database Concepts* for information about the memoptimize pool memory architecture

13

Tuning the Database Buffer Cache

This chapter describes how to tune the database buffer cache. If you are using automatic memory management to manage the database memory on your system, there is no need to manually tune the memory caches described in this chapter.

This chapter contains the following topics:

About the Database Buffer Cache

For many types of operations, Oracle Database uses the buffer cache to store data blocks read from disk. Oracle Database bypasses the buffer cache for particular operations, such as sorting and parallel reads.

To use the database buffer cache effectively, tune SQL statements for the application to avoid unnecessary resource consumption. To meet this goal, verify that frequently executed SQL statements and SQL statements that perform many buffer gets are well-tuned.

When using parallel query, consider configuring the database to use the database buffer cache instead of performing direct reads into the Program Global Area (PGA). This configuration may be appropriate when the system has a large amount of memory.

> **See Also:**
>
> - *Oracle Database SQL Tuning Guide* for information about tuning SQL statements
> - *Oracle Database VLDB and Partitioning Guide* for information about parallel execution

Configuring the Database Buffer Cache

When configuring a new database instance, it is impossible to know the correct size for the buffer cache. Typically, a database administrator makes a first estimate for the cache size, then runs a representative workload on the instance and examines the relevant statistics to see whether the cache is under-configured or over-configured.

This section describes how to configure the database buffer cache. If you are using automatic shared memory management to configure the Shared Global Area (SGA), there is no need to manually tune the database buffer cache as described in this section.

This section contains the following topics:

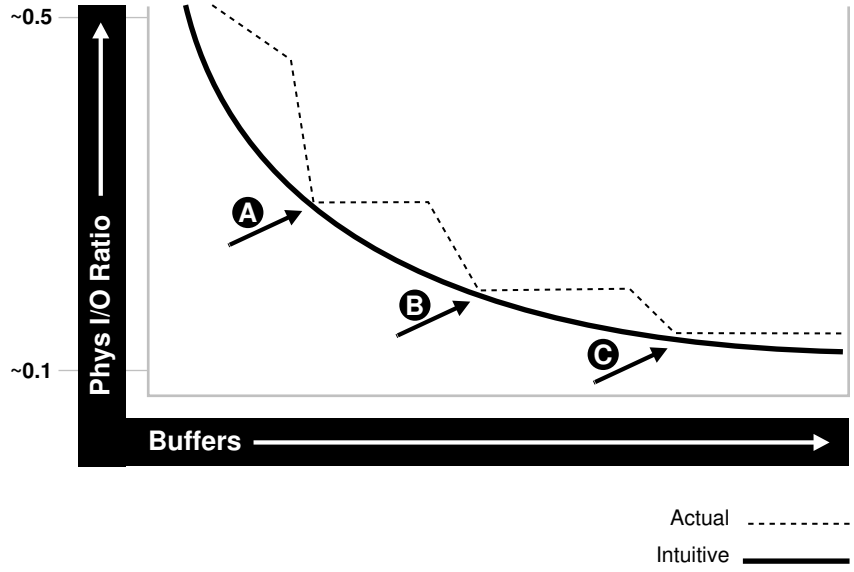
Using the V$DB_CACHE_ADVICE View

The `V$DB_CACHE_ADVICE` view shows the simulated miss rates for a range of potential buffer cache sizes. This view assists in cache sizing by providing information that predicts the number of physical reads for each potential cache size. The data also includes a physical read factor, which is a factor by which the current number of physical reads is estimated to change if the buffer cache is resized to a given value.

However, physical reads do not necessarily indicate disk reads in Oracle Database, because physical reads may be accomplished by reading from the file system cache. Hence, the relationship between successfully finding a block in the cache and the size of the cache is not always a smooth distribution. When sizing the buffer pool, avoid using additional buffers that do not contribute (or contribute very little) to the cache hit ratio.

The following figure illustrates the relationship between physical I/O ratio and buffer cache size.

Figure 13-1 Physical I/O Ratio and Buffer Cache Size



Examining the example illustrated in the above figure leads to the following observations:

- As the number of buffers increases, the physical I/O ratio decreases.
- The decrease in the physical I/O between points A and B and points B and C is not smooth, as indicated by the dotted line in the graph.
- The benefit from increasing buffers from point A to point B is considerably higher than from point B to point C.
- The benefit from increasing buffers decreases as the number of buffers increases.

There is some overhead associated with using this advisory view. When the advisory is enabled, there is a small increase in CPU usage, because additional bookkeeping is required. To reduce both the CPU and memory overhead associated with bookkeeping, Oracle Database uses sampling to gather cache advisory statistics. Sampling is not used if the number of buffers in a buffer pool is small to begin with.

To use the V$DB_CACHE_ADVICE view:

1. Set the value of the `DB_CACHE_ADVICE` initialization parameter to `ON`.

   This enables the advisory view. The `DB_CACHE_ADVICE` parameter is dynamic, so the advisory can be enabled and disabled dynamically to enable you to collect advisory data for a specific workload.

2. Run a representative workload on the database instance.

   Allow the workload to stabilize before querying the `V$DB_CACHE_ADVICE` view.

3. Query the `V$DB_CACHE_ADVICE` view.

The following example shows a query of this view that returns the predicted I/O requirement for the default buffer pool for various cache sizes.

```
COLUMN size_for_estimate          FORMAT 999,999,999,999 heading 'Cache Size (MB)'
COLUMN buffers_for_estimate       FORMAT 999,999,999 heading 'Buffers'
COLUMN estd_physical_read_factor  FORMAT 999.90 heading 'Estd Phys|Read Factor'
COLUMN estd_physical_reads        FORMAT 999,999,999 heading 'Estd Phys| Reads'

SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor,
       estd_physical_reads
FROM   V$DB_CACHE_ADVICE
WHERE  name = 'DEFAULT'
  AND  block_size = (SELECT value FROM V$PARAMETER WHERE name = 'db_block_size')
  AND  advice_status = 'ON';
```

The output of this query might look like the following:

```
                               Estd Phys    Estd Phys
 Cache Size (MB)      Buffers Read Factor        Reads
--------------- ------------ ----------- ------------
             30        3,802       18.70  192,317,943      10% of Current Size
             60        7,604       12.83  131,949,536
             91       11,406        7.38   75,865,861
            121       15,208        4.97   51,111,658
            152       19,010        3.64   37,460,786
            182       22,812        2.50   25,668,196
            212       26,614        1.74   17,850,847
            243       30,416        1.33   13,720,149
            273       34,218        1.13   11,583,180
            304       38,020        1.00   10,282,475      Current Size
            334       41,822         .93    9,515,878
```

```
364        45,624          .87     8,909,026
395        49,426          .83     8,495,039
424        53,228          .79     8,116,496
456        57,030          .76     7,824,764
486        60,832          .74     7,563,180
517        64,634          .71     7,311,729
547        68,436          .69     7,104,280
577        72,238          .67     6,895,122
608        76,040          .66     6,739,731      200% of Current Size
```

In this example, the output shows that if the cache was 212 MB instead of the current size of 304 MB, the estimated number of physical reads would increase by a factor of 1.74, or 74%. Hence, it is not advisable to decrease the cache size to 212MB.

However, increasing the cache size to 334MB may potentially decrease reads by a factor of .93, or 7%. If an additional 30MB memory is available on the system and the value of the `SGA_MAX_SIZE` parameter allows for the increment, it is advisable to increase the default buffer cache pool size to 334MB.

Calculating the Buffer Cache Hit Ratio

The buffer cache hit ratio calculates how often a requested block has been found in the buffer cache without requiring disk access. This ratio is computed using data selected from the `V$SYSSTAT` performance view. Use the buffer cache hit ratio to verify the physical I/O as predicted by the `V$DB_CACHE_ADVICE` view.

Table 13-1 lists the statistics from the `V$SYSSTAT` view used to calculate the buffer cache hit ratio.

Table 13-1 Statistics for Calculating the Buffer Cache Hit Ratio

Statistic	Description
`consistent gets from cache`	Number of times a consistent read was requested for a block from the buffer cache.
`db block gets from cache`	Number of times a `CURRENT` block was requested from the buffer cache.
`physical reads cache`	Total number of data blocks read from disk into buffer cache.

Example 13-1 shows a query of this view.

Example 13-1 Querying the V$SYSSTAT View

```
SELECT name, value
FROM V$SYSSTAT
WHERE name IN ('db block gets from cache', 'consistent gets from cache',
'physical reads cache');
```

In this example, the query is simplified by using values selected directly from the `V$SYSSTAT` view, rather than over an interval. It is recommended to calculate the delta of these statistics over an interval while the application is running, then use these delta values to determine the buffer cache hit ratio. For information about collecting statistics over an interval, see Automatic Performance Diagnostics .

Using the values from the output of this query, calculate the hit ratio for the buffer cache using the following formula:

```
1 - (('physical reads cache') / ('consistent gets from cache' +
'db block gets from cache'))
```

> **See Also:**
>
> *Oracle Database Reference* for information about the `V$SYSSTAT` view

Interpreting the Buffer Cache Hit Ratio

Before deciding whether to increase or decrease the buffer cache size, you should first examine the buffer cache hit ratio.

A low cache hit ratio does not necessarily imply that increasing the size of the buffer cache will benefit performance. Moreover, a high cache hit ratio may wrongly indicate that the buffer cache is adequately sized for the workload.

To interpret the buffer cache hit ratio, consider the following factors:

- Avoid repeated scanning of frequently accessed data by performing the processing in a single pass or by optimizing the SQL statement.

  Repeated scanning of the same large table or index can artificially inflate a low cache hit ratio. Examine frequently executed SQL statements with a large number of buffer gets, to ensure that the execution plans for these SQL statements are optimal.
- Avoid requerying the same data by caching frequently accessed data in the client program or middle tier.
- In large databases running OLTP applications, many rows are accessed only once (or never). Hence, there is no purpose in keeping the block in memory following its use.
- Do not continuously increase the buffer cache size.

  Continuous increases of the buffer cache size have no effect if the database is performing full table scans or operations that do not use the buffer cache.
- Consider poor hit ratios when large full table scans are occurring.

  Database blocks accessed during a long full table scan are placed on the tail end of the Least Recently Used (LRU) list and not on the head of the list. Therefore, the blocks age out faster than blocks read when performing indexed lookups or small table scans.

> **Note:**
>
> Short table scans are scans performed on tables under a certain size threshold. The definition of a small table is the maximum of 2% of the buffer cache or 20, whichever is bigger.

Increasing Memory Allocated to the Database Buffer Cache

If the cache hit ratio is low and your application is tuned to avoid performing full table scans, consider increasing the size of the buffer cache. If possible, resize the buffer pools dynamically, rather than shutting down the instance to perform this change.

To increase the size of the database buffer cache:

1. Set the value of the `DB_CACHE_ADVICE` initialization parameter to `ON`.
2. Allow the buffer cache statistics to stabilize.
3. Examine the advisory data in the `V$DB_CACHE_ADVICE` view to determine the next increment required to significantly decrease the amount of physical I/O performed, as described in "Using the V$DB_CACHE_ADVICE View".
4. If it is possible to allocate the extra memory required to the buffer cache without causing the system to page, then allocate this memory.
5. To increase the amount of memory allocated to the buffer cache, increase the value of the `DB_CACHE_SIZE` initialization parameter.

   The `DB_CACHE_SIZE` parameter specifies the size of the default cache for the database's standard block size. To create and use tablespaces with block sizes other than the database's standard block sizes (such as for transportable tablespaces), configure a separate cache for each block size used. Use the `DB_nK_CACHE_SIZE` parameter to configure the nonstandard block size needed (where `n` is 2, 4, 8, 16 or 32 and not the standard block size).

Note:

- The process of choosing a cache size is the same, regardless of whether the cache is the default standard block size cache, the `KEEP` or `RECYCLE` cache, or a nonstandard block size cache.
- When the cache is resized significantly (greater than 20%), the old cache advisory value is discarded and the cache advisory is set to the new size. Otherwise, the old cache advisory value is adjusted to the new size by the interpolation of existing values.

See Also:

For more information about the `DB_nK_CACHE_SIZE` parameter, see:

- *Oracle Database Administrator's Guide*
- *Oracle Database Reference*

Reducing Memory Allocated to the Database Buffer Cache

If the cache hit ratio is high, then the buffer cache is likely large enough to store the most frequently accessed data. If this is the case and memory is required for another memory structure, consider reducing the size of the buffer cache.

To reduce the size of the database buffer cache:

1. Examine the advisory data in the `V$DB_CACHE_ADVICE` view to determine if decreasing the size of the buffer cache will significantly increase the number of physical I/Os, as described in "Using the V$DB_CACHE_ADVICE View".
2. To reduce the amount of memory allocated to the buffer cache, decrease the value of the `DB_CACHE_SIZE` initialization parameter.

Configuring Multiple Buffer Pools

For most systems, a single default buffer pool is generally adequate. However, database administrators with detailed knowledge of an application's buffer pool may benefit from configuring multiple buffer pools.

For segments that have atypical access patterns, consider storing blocks from these segments in two separate buffer pools: the `KEEP` pool and the `RECYCLE` pool. A segment's access pattern may be atypical if it is constantly accessed (sometimes referred to as hot) or infrequently accessed (such as a large segment that is accessed by a batch job only once a day).

Using multiple buffer pools enables you to address these irregularities. You can use the `KEEP` pool to maintain frequently accessed segments in the buffer cache, and the `RECYCLE` pool to prevent objects from consuming unnecessary space in the buffer cache. When an object is associated with a buffer cache, all blocks from that object are placed in that cache. Oracle Database maintains a `DEFAULT` buffer pool for objects that are not assigned to a specific buffer pool. The default buffer pool size is determined by the `DB_CACHE_SIZE` initialization parameter. Each buffer pool uses the same LRU replacement policy. For example, if the `KEEP` pool is not large enough to store all of the segments allocated to it, then the oldest blocks age out of the cache.

By allocating objects to appropriate buffer pools, you can:

- Reduce or eliminate I/Os
- Isolate or limit an object to a separate cache

This section describes how to configure multiple buffer pools and contains the following topics:

- Considerations for Using Multiple Buffer Pools
- Using Multiple Buffer Pools
- Using the V$DB_CACHE_ADVICE View for Individual Buffer Pools
- Calculating the Buffer Pool Hit Ratio for Individual Buffer Pools
- Examining the Buffer Cache Usage Pattern
- Configuring the KEEP Pool
- Configuring the RECYCLE Pool

Considerations for Using Multiple Buffer Pools

When using multiple buffer pools, take the following considerations into account:

- Random Access to Large Segments

- Oracle Real Application Cluster Instances

Random Access to Large Segments

A problem may occur with an LRU aging method when a very large segment (compared to the size of the buffer cache) is accessed with a large or unbounded index range scan. Any single segment that accounts for a substantial portion (more than 10%) of nonsequential physical reads can be considered very large. Random reads to a large segment may cause buffers that contain data for other segments to be aged out of the cache. The large segment ends up consuming a large percentage of the buffer cache, but it does not benefit from the cache.

Very frequently accessed segments are not affected by large segment reads because their buffers are warmed frequently enough that they do not age out of the buffer cache. However, the problem affects warm segments that are not accessed frequently enough to survive the buffer aging caused by the large segment reads. There are three options for solving this problem:

- If the object accessed is an index, determine whether the index is selective. If not, tune the SQL statement to use a more selective index.
- If the SQL statement is tuned, move the large segment into a separate `RECYCLE` cache so it does not affect the other segments. The `RECYCLE` cache should be smaller than the `DEFAULT` buffer pool, and it should reuse buffers more quickly.
- Alternatively, consider moving the small, warm segments into a separate `KEEP` cache that is not used for large segments. Size the `KEEP` cache to minimize misses in the cache. You can make the response times for specific queries more predictable by storing the segments accessed by the queries in the `KEEP` cache to ensure that they do not age out.

Oracle Real Application Cluster Instances

In an Oracle Real Application Cluster (Oracle RAC) environment, consider creating multiple buffer pools for each database instance. It is not necessary to define the same set of buffer pools for each instance of the database. Among instances, the buffer pools can be different sizes or undefined. Tune each instance according to the application requirements for that instance.

Using Multiple Buffer Pools

To define a default buffer pool for an object, use the `BUFFER_POOL` keyword of the `STORAGE` clause. This clause is valid for the following SQL statements:

- `CREATE TABLE`
- `CREATE CLUSTER`
- `CREATE INDEX`
- `ALTER TABLE`
- `ALTER CLUSTER`
- `ALTER INDEX`

After a buffer pool is defined, all subsequent blocks read for the object are placed in that pool. If a buffer pool is defined for a partitioned table or index, then each partition

of the object inherits the buffer pool from the table or index definition, unless if it is overridden by a specific buffer pool.

When the buffer pool of an object is changed using the `ALTER` statement, all buffers currently containing blocks of the altered segment remain in the buffer pool they were in before the `ALTER` statement. Newly loaded blocks and any blocks that age out and are reloaded are placed into the new buffer pool.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about specifying `BUFFER_POOL` in the `STORAGE` clause

Using the V$DB_CACHE_ADVICE View for Individual Buffer Pools

As with the default buffer pool, you can use `V$DB_CACHE_ADVICE` view to assist in cache sizing of other pools. After estimating the initial cache size and running a representative workload, query the `V$DB_CACHE_ADVICE` view for the pool you want to use.

For more information about using the `V$DB_CACHE_ADVICE` view, see "Using the V$DB_CACHE_ADVICE View".

Example 13-2 shows a query of this view that queries data from the `KEEP` pool:

Example 13-2 Querying the V$DB_CACHE_ADVICE View for the KEEP Pool

```
SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor,
       estd_physical_reads
  FROM V$DB_CACHE_ADVICE
 WHERE name = 'KEEP'
   AND block_size = (SELECT value FROM V$PARAMETER WHERE name = 'db_block_size')
   AND advice_status = 'ON';
```

Calculating the Buffer Pool Hit Ratio for Individual Buffer Pools

The data in the `V$SYSSTAT` view reflects the logical and physical reads for all buffer pools within one set of statistics. To determine the hit ratio for the buffer pools individually, query the `V$BUFFER_POOL_STATISTICS` view. This view maintains statistics on the number of logical reads and writes for each pool.

> **See Also:**
>
> - "Calculating the Buffer Cache Hit Ratio" for more information about calculating hit ratios
> - *Oracle Database Reference* for more information about `V$BUFFER_POOL_STATISTICS` view

The following query calculates the hit ratio using the `V$BUFFER_POOL_STATISTICS` view.

Example 13-3 Querying the V$BUFFER_POOL_STATISTICS View

```
SELECT name, physical_reads, db_block_gets, consistent_gets,
       1 - (physical_reads / (db_block_gets + consistent_gets)) "Hit Ratio"
  FROM V$BUFFER_POOL_STATISTICS;
```

Examining the Buffer Cache Usage Pattern

The V$BH view shows the data object ID of all blocks that currently reside in the SGA. To determine which segments have many buffers in the pool, use this view to examine the buffer cache usage pattern. You can either examine the buffer cache usage pattern for all segments or a specific segment, as described in the following sections:

- Examining the Buffer Cache Usage Pattern for All Segments
- Examining the Buffer Cache Usage Pattern for a Specific Segment

Examining the Buffer Cache Usage Pattern for All Segments

One method to determine which segments have many buffers in the pool is to query the number of blocks for all segments that reside in the buffer cache at a given time. Depending on buffer cache size, this might require a lot of sort space.

Example 13-4 shows a query that counts the number of blocks for all segments.

Example 13-4 Querying the Number of Blocks for All Segments

```
COLUMN object_name FORMAT A40
COLUMN number_of_blocks FORMAT 999,999,999,999

SELECT o.object_name, COUNT(*) number_of_blocks
  FROM DBA_OBJECTS o, V$BH bh
 WHERE o.data_object_id = bh.OBJD
   AND o.owner != 'SYS'
 GROUP BY o.object_Name
 ORDER BY COUNT(*);
```

The output of this query might look like the following:

```
OBJECT_NAME                              NUMBER_OF_BLOCKS
---------------------------------------- ----------------
OA_PREF_UNIQ_KEY                                        1
SYS_C002651                                             1
..
DS_PERSON                                              78
OM_EXT_HEADER                                         701
OM_SHELL                                            1,765
OM_HEADER                                           5,826
OM_INSTANCE                                        12,644
```

Examining the Buffer Cache Usage Pattern for a Specific Segment

Another method to determine which segments have many buffers in the pool is to calculate the percentage of the buffer cache used by an individual object at a given time.

To calculate the percentage of the buffer cache used by an individual object:

1. Find the Oracle Database internal object number of the segment by querying the `DBA_OBJECTS` view:

   ```
   SELECT data_object_id, object_type
   FROM DBA_OBJECTS
   WHERE object_name = UPPER('segment_name');
   ```

   Because two objects can have the same name (if they are different types of objects), use the `OBJECT_TYPE` column to identify the object of interest.

2. Find the number of buffers in the buffer cache for *`SEGMENT_NAME`*:

   ```
   SELECT COUNT(*) buffers
   FROM V$BH
   WHERE objd = data_object_id_value;
   ```

   For *`data_object_id_value`*, use the value of `DATA_OBJECT_ID` from the previous step.

3. Find the number of buffers in the database instance:

   ```
   SELECT name, block_size, SUM(buffers)
   FROM V$BUFFER_POOL
   GROUP BY name, block_size
   HAVING SUM(buffers) > 0;
   ```

4. Calculate the ratio of buffers to total buffers to obtain the percentage of the cache currently used by *`SEGMENT_NAME`*:

   ```
   % cache used by segment_name = [buffers(Step2)/total buffers(Step3)]
   ```

> **Note:**
>
> This method works only for a single segment. For a partitioned object, run the query for each partition.

Configuring the KEEP Pool

The purpose of the `KEEP` buffer pool is to retain objects in memory, thus avoiding I/O operations. Each object kept in memory results in a trade-off. It is more beneficial to keep frequently-accessed blocks in the cache. Avoid retaining infrequently-used blocks in the cache, as this results in less space for other, more active blocks

If there are certain segments in your application that are referenced frequently, then consider storing the blocks from those segments in the `KEEP` buffer pool. Typical segments that are kept in the `KEEP` pool are small, frequently-used reference tables. To determine which tables are candidates, check the number of blocks from candidate tables by querying the `V$BH` view, as described in "Examining the Buffer Cache Usage Pattern".

To configure the KEEP pool:

1. Compute an approximate size for the `KEEP` buffer pool.

   The size of the `KEEP` buffer pool depends on the objects to be kept in the buffer cache. To estimate its size, add the blocks used by all objects assigned to this pool.

If you gathered statistics on the segments, query `DBA_TABLES.BLOCKS` and `DBA_TABLES.EMPTY_BLOCKS` to determine the number of blocks used.

2. Taking two snapshots of system performance at different times.

   Query data from the `KEEP` pool for each snapshot using the `V$DB_CACHE_ADVICE` view, as described in "Using the V$DB_CACHE_ADVICE View for Individual Buffer Pools".

3. Subtract the more recent values for `physical reads`, `block gets`, and `consistent gets` from the older values, and use the results to calculate the hit ratio.

   A buffer pool hit ratio of 100% may not be optimal. Oftentimes, you can decrease the size of the `KEEP` buffer pool and still maintain a sufficiently high hit ratio. Allocate blocks removed from the `KEEP` buffer pool to other buffer pools.

4. Allocate memory to the `KEEP` buffer pool by setting the value of the `DB_KEEP_CACHE_SIZE` parameter to the required size.

   The memory for the `KEEP` pool is not a subset of the default pool.

> **Note:**
>
> If an object grows in size, then it might no longer fit in the `KEEP` buffer pool. You will begin to lose blocks out of the cache.

Configuring the RECYCLE Pool

You can configure a `RECYCLE` buffer pool for blocks belonging to those segments that you do not want to keep in memory. The purpose of the `RECYCLE` pool is to retain segments that are scanned rarely or are not referenced frequently. If an application randomly accesses the blocks of a very large object, then it is unlikely for a block stored in the buffer pool to be reused before it is aged out. This is true regardless of the size of the buffer pool (given the constraint of the amount of available physical memory). Consequently, the object's blocks do not need to be cached; the cache buffers can be allocated to other objects.

Do not discard blocks from memory too quickly. If the buffer pool is too small, then blocks can age out of the cache before the transaction or SQL statement completes its execution. For example, an application might select a value from a table, use the value to process some data, and then update the record. If the block is removed from the cache after the `SELECT` statement, then it must be read from disk again to perform the update. The block should be retained for the duration of the user transaction.

To configure the RECYCLE POOL:

- Allocate memory to the `RECYCLE` buffer pool by setting the value of the `DB_RECYCLE_CACHE_SIZE` parameter to the required size.

  The memory for the `RECYCLE` pool is not a subset of the default pool.

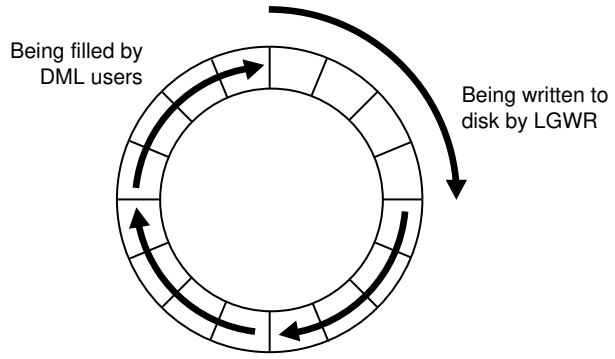
Configuring the Redo Log Buffer

Server processes making changes to data blocks in the buffer cache generate redo data into the log buffer. The log writer process (LGWR) begins writing to copy entries from the redo log buffer to the online redo log if any of the following conditions are true:

- The redo log buffer becomes at least one-third full
- LGWR is posted by a server process performing a `COMMIT` or `ROLLBACK`
- A database writer process (DBWR) posts LGWR to do so

When LGWR writes redo entries from the redo log buffer to a redo log file or disk, user processes can copy new entries over the entries in memory that are written to disk, as illustrated in the following figure.

Figure 13-2 Redo Log Buffer



LGWR attempts to write fast enough to ensure that space is available in the redo log buffer for new entries, even if it is frequently accessed. Having a larger redo log buffer makes it more likely that there is space for new entries, and also enables LGWR to efficiently process redo records. On a system with large updates, if the redo log buffer is too small, LGWR will continuously flush redo to disk so that it remains two-thirds empty.

On systems with fast processors and relatively slow disks, the processors might be filling the rest of the redo log buffer in the time it takes the redo log writer to move a portion of the redo log buffer to disk. In this situation, a larger redo log buffer can temporarily mask the effects of slower disks. Alternatively, consider either improving:

- The checkpointing or archiving process
- The performance of LGWR by moving all online logs to fast raw devices

To improve the performance of the redo log buffer, ensure that you are:

- Batching commit operations for batch jobs, so that LGWR is able to write redo log entries efficiently
- Using `NOLOGGING` operations when loading large quantities of data

This section describes how to configure the redo log buffer and contains the following topics:

Sizing the Redo Log Buffer

The default size of the redo log buffer is calculated as follows:

```
MAX(0.5M, (128K * number of cpus))
```

Applications that insert, modify, or delete large volumes of data may require changing the default size of the redo log buffer. Oracle recommends setting the redo log buffer size to minimum of 8 MB. Set it to a minimum of 64 MB for databases using flashback functionality and having 4GB or higher SGAs. Set it to a minimum of 256 MB if you are using Oracle Data Guard with asynchronous redo transport and have a high redo generation rate.

To determine if the size of the redo log buffer is too small, monitor the redo log buffer statistics, as described in "Using Redo Log Buffer Statistics". You can also check if the `log buffer space` wait event is a significant factor in the wait time for the database instance. If it is not, then the log buffer size is most likely adequately-sized.

To size the redo log buffer:

- Set the size of the redo log buffer by setting the value of the `LOG_BUFFER` initialization parameter to the required size.

  The value of this parameter is expressed in bytes.

> **Note:**
>
> The size of the redo log buffer cannot be modified after instance startup.

Using Redo Log Buffer Statistics

The `REDO BUFFER ALLOCATION RETRIES` statistic reflects the number of times a user process waits for space in the redo log buffer. This statistic can be queried using the `V$SYSSTAT` performance view.

You should monitor the `redo buffer allocation retries` statistic over a period while the application is running. The value of this statistic should be near zero over an interval. If this value increases consistently, then it means user processes had to wait for space in the redo log buffer to become available. The wait can be caused by the redo log buffer being too small or by checkpointing. In this case, consider one of the following options:

- Increase the size of the redo log buffer, as described in "Sizing the Redo Log Buffer"
- Improve the checkpointing or archiving process

Example 13-5 shows a query of the `V$SYSSTAT` view for this statistic.

Example 13-5 Querying the V$SYSSTAT View

```
SELECT name, value
  FROM V$SYSSTAT
 WHERE name = 'redo buffer allocation retries';
```

Configuring the Database Caching Mode

Starting with Oracle Database 12*c* Release 1 (12.1.0.2), there are two database caching modes: the default database caching mode used in previous versions of Oracle Database, and the force full database caching mode that is new to this release. In default caching mode, Oracle Database does not always cache the underlying data when a user queries a large table. In force full database caching mode, Oracle Database assumes that the buffer cache is large enough to cache the full database and tries to cache all the blocks that are accessed by queries.

This section contains the following topics:

- Default Database Caching Mode
- Force Full Database Caching Mode
- Determining When to Use Force Full Database Caching Mode
- Verifying the Database Caching Mode

> **Note:**
>
> Force full database caching mode is available starting with Oracle Database 12*c* Release 1 (12.1.0.2).

Default Database Caching Mode

By default, Oracle Database uses the default database caching mode when performing full table scans. In default caching mode, Oracle Database does not always cache the underlying data when a user queries a large table, because doing so might remove more useful data from the buffer cache.

If the Oracle Database instance determines that there is enough space to cache the full database in the buffer cache and that it would be beneficial to do so, then the instance automatically caches the full database in the buffer cache.

If the Oracle Database instance determines that there is not enough space to cache the full database in the buffer cache, then:

- Smaller tables are loaded into memory only when the table size is less than 2 percent of the buffer cache size.
- For medium tables, Oracle Database analyzes the interval between the last table scan and the aging timestamp of the buffer cache. If the size of the table reused in the last table scan is greater than the remaining buffer cache size, then the table is cached.
- Large tables are typically not loaded into memory, unless if you explicitly declare the table for the KEEP buffer pool.

> **Note:**
>
> In default caching mode, Oracle Database instance does not cache `NOCACHE` LOBs in the buffer cache.

> **See Also:**
>
> *Oracle Database Concepts* for information about the default database caching mode

Force Full Database Caching Mode

As more memory is added to a database, buffer cache sizes may continually grow. In some cases, the size of the buffer cache may become so large that the entire database can fit into memory. The ability to cache an entire database in memory can drastically improve database performance when performing full table scans or accessing LOBs.

In force full database caching mode, Oracle Database caches the entire database in memory when the size of the database is smaller than the database buffer cache size. All data files, including `NOCACHE` LOBs and LOBS that use SecureFiles, are loaded into the buffer cache as they are being accessed.

> **See Also:**
>
> - *Oracle Database Concepts*
> - *Oracle Database Administrator's Guide*

Determining When to Use Force Full Database Caching Mode

To improve database performance for table scans and LOB data access, especially for workloads that are limited by I/O throughput or response time, consider using force full database caching mode whenever the size of the database buffer cache is greater than the size of the database.

Consider using force full database caching mode in the following situations:

- The logical database size (or actual used space) is smaller than the individual buffer cache of each database instance in an Oracle RAC environment. This is applicable for non-Oracle RAC database as well.
- The logical database size is smaller than 80% of the combined buffer cache sizes of all the database instances for well-partitioned workloads (by instance access) in an Oracle RAC environment.
- The database uses SGA_TARGET or MEMORY_TARGET.

- The `NOCACHE` LOBs need to be cached. The `NOCACHE` LOBs are never cached unless force full database caching is used.

For the first three situations, you should monitor the system performance periodically to verify that the performance figures are according to your expectations.

When one Oracle RAC database instance uses force full database caching mode, then all the other database instances in the Oracle RAC environment will also use force full database caching mode.

In a multitenant environment, force full database caching mode applies to the entire container database (CDB), including all of its pluggable databases (PDBs).

Verifying the Database Caching Mode

By default, Oracle Database runs in the default database caching mode.

To verify if force full database caching mode is enabled:

- Query the `V$DATABASE` view as shown:

```
SELECT FORCE_FULL_DB_CACHING FROM V$DATABASE;
```

If the query returns a value of `YES`, then force full database caching mode is enabled on the database. If the query returns a value of `NO`, then force full database caching mode is disabled and the database is in default database caching mode.

> **Note:**
>
> To enable force full database caching mode, use the following `ALTER DATABASE` command:
>
> ```
> ALTER DATABASE FORCE FULL DATABASE CACHING;
> ```

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about enabling and disabling force full database caching mode
> - *Oracle Database Reference* for more information about the `V$DATABASE` view

14

Tuning the Shared Pool and the Large Pool

This chapter describes how to tune the shared pool and the large pool. If you are using automatic memory management to manage the database memory on your system, or automatic shared memory management to configure the Shared Global Area (SGA), there is no need to manually tune the shared pool and the large pool as described in this chapter.

This chapter contains the following topics:

About the Shared Pool

Oracle Database uses the shared pool to cache many different types of data. Cached data includes the textual and executable forms of PL/SQL blocks and SQL statements, dictionary cache data, result cache data, and other data.

This section describes the shared pool and contains the following topics:

Benefits of Using the Shared Pool

Proper use and sizing of the shared pool can reduce resource consumption in at least four ways:

- If the SQL statement is in the shared pool, parse overhead is avoided, resulting in reduced CPU resources on the system and elapsed time for the end user.
- Latching resource usage is significantly reduced, resulting in greater scalability.
- Shared pool memory requirements are reduced, because all applications use the same pool of SQL statements and dictionary resources.
- I/O is reduced, because dictionary elements that are in the shared pool do not require disk access.

Shared Pool Concepts

The main components of the shared pool include:

- Library cache

  The library cache stores the executable (parsed or compiled) form of recently referenced SQL and PL/SQL code.

- Data dictionary cache

  The data dictionary cache stores data referenced from the data dictionary.

- Server result cache (depending on the configuration)

  The server result cache is an optional cache that stores query and PL/SQL function results within the shared pool. For information about the server result cache, see "About the Result Cache".

Many of the caches in the shared pool—including the library cache and the dictionary cache—automatically increase or decrease in size, as needed. Old entries are aged out to accommodate new entries when the shared pool runs out of space.

A cache miss on the library cache or data dictionary cache is more expensive than a miss on the buffer cache. For this reason, the shared pool should be sized to ensure that frequently-used data is cached.

Several features require large memory allocations in the shared pool, such as shared server, parallel query, or Recovery Manager. Oracle recommends using a separate memory area—the large pool—to segregate the System Global Area (SGA) memory used by these features.

Allocation of memory from the shared pool is performed in chunks. This chunking enables large objects (over 5 KB) to be loaded into the cache without requiring a single contiguous area. In this way, the database reduces the possibility of running out of contiguous memory due to fragmentation.

Java, PL/SQL, or SQL cursors may sometimes make allocations out of the shared pool that are larger than 5 KB. To enable these allocations to occur more efficiently, Oracle Database segregates a small amount of the shared pool. The segregated memory, called the reserved pool, is used if the shared pool runs out of space.

The following sections provide more details about the main components of the shared pool:

- Library Cache Concepts
- Data Dictionary Cache Concepts
- SQL Sharing Criteria

Library Cache Concepts

The library cache stores executable forms of SQL cursors, PL/SQL programs, and Java classes, which are collectively referred to as the application code. This section focuses on tuning as it relates to the application code.

When the application code is executed, Oracle Database attempts to reuse existing code if it has been executed previously and can be shared. If the parsed representation of the SQL statement exists in the library cache and it can be shared, then the database reuses the existing code. This is known as a soft parse, or a library cache hit. If Oracle Database cannot use the existing code, then the database must build a new executable version of the application code. This is known as a hard parse, or a library cache miss. For information about when SQL and PL/SQL statements can be shared, see "SQL Sharing Criteria".

In order to perform a hard parse, Oracle Database uses more resources than during a soft parse. Resources used for a soft parse include CPU and library cache latch gets. Resources required for a hard parse include additional CPU, library cache latch gets,

and shared pool latch gets. A hard parse may occur on either the parse step or the execute step when processing a SQL statement.

When an application makes a parse call for a SQL statement, if the parsed representation of the statement does not exist in the library cache, then Oracle Database parses the statement and stores the parsed form in the shared pool. To reduce library cache misses on parse calls, ensure that all sharable SQL statements are stored in the shared pool whenever possible.

When an application makes an execute call for a SQL statement, if the executable portion of the SQL statement is aged out (or deallocated) from the library cache to make room for another statement, then Oracle Database implicitly reparses the statement to create a new shared SQL area for it, and executes the statement. This also results in a hard parse. To reduce library cache misses on execution calls, allocate more memory to the library cache.

For more information about hard and soft parsing, see "SQL Execution Efficiency".

Data Dictionary Cache Concepts

Information stored in the data dictionary cache includes:

- Usernames
- Segment information
- Profile data
- Tablespace information
- Sequence numbers

The data dictionary cache also stores descriptive information, or metadata, about schema objects. Oracle Database uses this metadata when parsing SQL cursors or during the compilation of PL/SQL programs.

SQL Sharing Criteria

Oracle Database automatically determines whether a SQL statement or PL/SQL block being issued is identical to another statement currently in the shared pool.

To compare the text of the SQL statement to the existing SQL statements in the shared pool, Oracle Database performs the following steps:

1. The text of the SQL statement is hashed.

   If there is no matching hash value, then the SQL statement does not currently exist in the shared pool, and a hard parse is performed.

2. If there is a matching hash value for an existing SQL statement in the shared pool, then the text of the matched statement is compared to the text of the hashed statement to verify if they are identical.

   The text of the SQL statements or PL/SQL blocks must be identical, character for character, including spaces, case, and comments. For example, the following statements cannot use the same shared SQL area:

   ```
   SELECT * FROM employees;
   SELECT * FROM Employees;
   SELECT *  FROM employees;
   ```

   Also, SQL statements that differ only in literals cannot use the same shared SQL area. For example, the following statements do not resolve to the same SQL area:

```
SELECT count(1) FROM employees WHERE manager_id = 121;
SELECT count(1) FROM employees WHERE manager_id = 247;
```

The only exception to this rule is when the `CURSOR_SHARING` parameter is set to `FORCE`, in which case similar statements can share SQL areas. For information about the costs and benefits involved in cursor sharing, see "Sharing Cursors".

3. The objects referenced in the issued statement are compared to the referenced objects of all existing statements in the shared pool to ensure that they are identical.

   References to schema objects in the SQL statements or PL/SQL blocks must resolve to the same object in the same schema. For example, if two users each issue the following SQL statement but they each have their own `employees` table, then this statement is not considered identical, because the statement references different tables for each user:

   ```
   SELECT * FROM employees;
   ```

4. Bind variables in the SQL statements must match in name, data type, and length.

   For example, the following statements cannot use the same shared SQL area, because the bind variable names are different:

   ```
   SELECT * FROM employees WHERE department_id = :department_id;
   SELECT * FROM employees WHERE department_id = :dept_id;
   ```

   Many Oracle products, such as Oracle Forms and the precompilers, convert the SQL before passing statements to the database. Characters are uniformly changed to uppercase, white space is compressed, and bind variables are renamed so that a consistent set of SQL statements is produced.

5. The session's environment must be identical.

   For example, SQL statements must be optimized using the same optimization goal.

> **See Also:**
>
> *Oracle Database Reference* for information about the `CURSOR_SHARING` initialization parameter

Using the Shared Pool

An important purpose of the shared pool is to cache the executable versions of SQL and PL/SQL statements. This enables multiple executions of the same SQL or PL/SQL code to be performed without the resources required for a hard parse, which results in significant reductions in CPU, memory, and latch usage.

The shared pool is also able to support unshared SQL in data warehousing applications, which execute low-concurrency, high-resource SQL statements. In this situation, using unshared SQL with literal values is recommended. Using literal values rather than bind variables enables the optimizer to make good column selectivity estimates, thus providing an optimal data access plan.

In a high-currency online transaction processing (OLTP) system, efficient use of the shared pool significantly reduces the probability of parse-related application scalability

issues. There are several ways to ensure efficient use of the shared pool and related resources in an OLTP system:

- Use Shared Cursors
- Use Single-User Logon and Qualified Table Reference
- Use PL/SQL
- Avoid Performing DDL Operations
- Cache Sequence Numbers
- Control Cursor Access
- Maintain Persistent Connections

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide* for information about impact of parallel query execution on the shared pool

Use Shared Cursors

Reuse of shared SQL for multiple users running the same application avoids hard parsing. Soft parses provide a significant reduction in the use of resources, such as the shared pool and library cache latches.

To use shared cursors:

- Use bind variables instead of literals in SQL statements whenever possible.

  For example, the following two SQL statements cannot use the same shared area because they do not match character for character:

  ```
  SELECT employee_id FROM employees WHERE department_id = 10;
  SELECT employee_id FROM employees WHERE department_id = 20;
  ```

  Replacing the literals with a bind variable results in only one SQL statement which can be executed twice:

  ```
  SELECT employee_id FROM employees WHERE department_id = :dept_id;
  ```

  For existing applications where rewriting the code to use bind variables is not possible, use the `CURSOR_SHARING` initialization parameter to avoid some of the hard parse overhead, as described in "Sharing Cursors".
- Avoid application designs that result in large numbers of users issuing dynamic, unshared SQL statements.

  Typically, the majority of data required by most users can be satisfied using preset queries. Use dynamic SQL where such functionality is required.
- Ensure that users of the application do not change the optimization approach and goal for their individual sessions.
- Establish the following policies for application developers:
  - Standardize naming conventions for bind variables and spacing conventions for SQL statements and PL/SQL blocks.

  - Consider using stored procedures whenever possible.

    Multiple users issuing the same stored procedure use the same shared PL/SQL area automatically. Because stored procedures are stored in a parsed form, their use reduces run-time parsing.

- For SQL statements which are identical but are not being shared, query the `V$SQL_SHARED_CURSOR` view to determine why the cursors are not shared.

  This includes optimizer settings and bind variable mismatches.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for more information about cursor sharing

Use Single-User Logon and Qualified Table Reference

In large OLTP systems where users log in to the database with their own user logon, qualifying the segment owner explicitly instead of using public synonyms significantly reduces the number of entries in the dictionary cache.

An alternative to qualifying table names is to connect to the database through a single user logon, rather than individual user logons. User-level validation can take place locally on the middle tier.

Use PL/SQL

Using stored PL/SQL packages can overcome many of the scalability issues for systems with thousands of users, each with individual user logon and public synonyms. This is because a package is executed as the owner, rather than the caller, which reduces the dictionary cache load considerably.

> **Note:**
>
> Oracle encourages the use of definer's rights packages to overcome scalability issues. The benefits of reduced dictionary cache load are not as great with invoker's rights packages.

Avoid Performing DDL Operations

Avoid performing DDL operations on high-usage segments during peak hours. Performing DDL operations on these segments often results in the dependent SQL being invalidated and reparsed in a later execution.

Cache Sequence Numbers

Allocating sufficient cache space for frequently updated sequence numbers significantly reduces the frequency of dictionary cache locks, which improves scalability.

To configure the number of cache entries for each sequence:

- Use the `CACHE` keyword in the `CREATE SEQUENCE` or `ALTER SEQUENCE` statement.

Control Cursor Access

Depending on your application tool, you can control how frequently the application performs parse calls by controlling cursor access.

The frequency with which the application either closes cursors or reuses existing cursors for new SQL statements affects the amount of memory used by a session, and often the amount of parsing performed by that session. An application that closes cursors or reuses cursors (for a different SQL statement) does not require as much session memory as an application that keeps cursors open. Conversely, that same application may need to perform more parse calls, using more CPU and database resources

Cursors associated with SQL statements that are not executed frequently can be closed or reused for other statements, because the likelihood of re-executing (and reparsing) that statement is low. Extra parse calls are required when a cursor containing a SQL statement that will be re-executed is closed or reused for another statement. Had the cursor remained open, it may have been reused without the overhead of issuing a parse call.

The ways in which you control cursor access depends on your application development tool. This section describes the methods used for Oracle Database tools:

- Controlling Cursor Access Using OCI
- Controlling Cursor Access Using Oracle Precompilers
- Controlling Cursor Access Using SQLJ
- Controlling Cursor Access Using JDBC
- Controlling Cursor Access Using Oracle Forms

> **See Also:**
>
> The tool-specific documentation for information about each tool

Controlling Cursor Access Using OCI

When using Oracle Call Interface (OCI), do not close and reopen cursors that you will be re-executing. Instead, leave the cursors open, and change the literal values in the bind variables before execution.

Avoid reusing statement handles for new SQL statements when the existing SQL statement will be re-executed in the future.

Controlling Cursor Access Using Oracle Precompilers

When using the Oracle precompilers, you can control when cursors are closed by setting precompiler clauses. In Oracle mode, the clauses are as follow:

- `HOLD_CURSOR = YES`

- `RELEASE_CURSOR = NO`
- `MAXOPENCURSORS = desired_value`

The precompiler clauses can be specified on the precompiler command line or within the precompiler program. Oracle recommends that you not use ANSI mode, in which the values of `HOLD_CURSOR` and `RELEASE_CURSOR` are switched.

> **See Also:**
>
> Your language's precompiler manual for information about these clauses

Controlling Cursor Access Using SQLJ

Prepare the SQL statement, then re-execute the statement with the new values for the bind variables. The cursor stays open for the duration of the session.

> **Note:**
>
> Starting with Oracle Database 12*c* Release 2 (12.2), server-side SQLJ code is not supported, that is, you cannot use SQLJ code inside stored procedures, functions, and triggers.

Controlling Cursor Access Using JDBC

Avoid closing cursors if they will be re-executed, because the new literal values can be bound to the cursor for re-execution. Alternatively, JDBC provides a SQL statement cache within the JDBC client using the `setStmtCacheSize()` method. Using this method, JDBC creates a SQL statement cache that is local to the JDBC program.

> **See Also:**
>
> *Oracle Database JDBC Developer's Guide* for information about using the JDBC SQL statement cache

Controlling Cursor Access Using Oracle Forms

With Oracle Forms, it is possible to control some aspects of cursor access at run time, the trigger level, or the form level.

Maintain Persistent Connections

Large OLTP applications with middle tiers should maintain connections, instead of connecting and disconnecting for each database request. Maintaining persistent connections saves CPU resources and database resources, such as latches.

Configuring the Shared Pool

This section describes how to configure the shared pool and contains the following topics:

Sizing the Shared Pool

When configuring a new database instance, it is difficult to know the correct size for the shared pool cache. Typically, a DBA makes a first estimate for the cache size, then runs a representative workload on the instance, and examines the relevant statistics to see whether the cache is under-configured or over-configured.

For most OLTP applications, shared pool size is an important factor in application performance. Shared pool size is less important for applications that issue a very limited number of discrete SQL statements, such as decision support systems (DSS).

If the shared pool is too small, then extra resources are used to manage the limited amount of available space. This consumes CPU and latching resources, and causes contention. Ideally, the shared pool should be just large enough to cache frequently-accessed objects. Having a significant amount of free memory in the shared pool is a waste of memory. When examining the statistics after the database has been running, ensure that none of these mistakes are present in the workload.

This section describes how to size the shared pool and contains the following topics:

Using Library Cache Statistics

When sizing the shared pool, the goal is to cache SQL statements that are executed multiple times in the library cache without over-allocating memory. To accomplish this goal, examine the following library cache statistics:

- `RELOADS`

  The `RELOADS` column in the `V$LIBRARYCACHE` view shows the amount of reloading (or reparsing) of a previously-cached SQL statement that aged out of the cache. If the application reuses SQL effectively and runs on a system with an optimal shared pool size, this statistic should have a value near zero.

- `INVALIDATIONS`

The INVALIDATIONS column in V$LIBRARYCACHE view shows the number of times library cache data was invalidated and had to be reparsed. This statistic should have a value near zero, especially on OLTP systems during peak loads. This means SQL statements that can be shared were invalidated by some operation (such as a DDL).

- Library cache hit ratio

  The library cache hit ratio is a broad indicator of the library cache health. This value should be considered along with the other statistics, such as the rate of hard parsing and if there is any shared pool or library cache latch contention.

- Amount of free memory in the shared pool

  To view the amount of free memory in the shared pool, query the V$SGASTAT performance view. Ideally, free memory should be as low as possible, without causing any reparsing on the system.

The following sections describe how to view and examine these library cache statistics:

- Using the V$LIBRARYCACHE View
- Calculating the Library Cache Hit Ratio
- Viewing the Amount of Free Memory in the Shared Pool

Using the V$LIBRARYCACHE View

Use the V$LIBRARYCACHE view to monitor statistics that reflect library cache activity. These statistics reflect all library cache activity after the most recent database instance startup.

Each row in this view contains statistics for one type of item kept in the library cache. The item described by each row is identified by the value of the NAMESPACE column. Rows with the following NAMESPACE values reflect library cache activity for SQL statements and PL/SQL blocks:

- SQL AREA
- TABLE/PROCEDURE
- BODY
- TRIGGER

Rows with other NAMESPACE values reflect library cache activity for object definitions that Oracle Database uses for dependency maintenance.

Example 14-1 shows a query of this view to examine each namespace individually.

Example 14-1 Querying the V$LIBRARYCACHE View

```
SELECT namespace, pins, pinhits, reloads, invalidations
  FROM V$LIBRARYCACHE
 ORDER BY namespace;
```

The output of this query might look like the following:

```
NAMESPACE             PINS    PINHITS    RELOADS INVALIDATIONS
--------------- ---------- ---------- ---------- -------------
BODY                  8870       8819          0             0
CLUSTER                393        380          0             0
```

```
INDEX                    29          0          0          0
OBJECT                    0          0          0          0
PIPE                  55265      55263          0          0
SQL AREA           21536413   21520516      11204          2
TABLE/PROCEDURE    10775684   10774401          0          0
TRIGGER                1852       1844          0          0
```

In this example, the output shows that:

- For the `SQL AREA` namespace, there are 21,536,413 executions.
- 11,204 of these executions resulted in a library cache miss, requiring the database to implicitly reparse a statement or block, or reload an object definition because it aged out of the library cache.
- SQL statements are invalidated twice, again causing library cache misses.

> **Note:**
>
> This query returns data from instance startup. Using statistics gathered over an interval instead may better identify the problem. For information about gathering information over an interval, see Automatic Performance Diagnostics .

> **See Also:**
>
> *Oracle Database Reference* for information about the `V$LIBRARYCACHE` view

Calculating the Library Cache Hit Ratio

To calculate the library cache hit ratio, use the following formula:

```
Library Cache Hit Ratio = sum(pinhits) / sum(pins)
```

Applying the library cache hit ratio formula to Example 14-1 results in the following library cache hit ratio:

```
SUM(PINHITS)/SUM(PINS)
----------------------
            .999466248
```

In this example, the hit percentage is about 99.94%, which means that only .06% of executions resulted in reparsing.

Viewing the Amount of Free Memory in the Shared Pool

The amount of free memory in the shared pool is reported in the `V$SGASTAT` view.

Example 14-2 shows a query of this view.

Example 14-2 Querying the V$SGASTAT View

```
SELECT *
  FROM V$SGASTAT
```

```
 WHERE name = 'free memory'
   AND pool = 'shared pool';
```

The output of this query might look like the following:

```
POOL        NAME                           BYTES
----------- -------------------------- ----------
shared pool free memory                  4928280
```

If free memory is always available in the shared pool, then increasing its size offers little or no benefit. Yet, just because the shared pool is full does not necessarily mean there is a problem. It may be indicative of a well-configured system.

Using Shared Pool Advisory Statistics

The amount of memory available for the library cache can drastically affect the parse rate of Oracle Database. To help you correctly size the library cache, Oracle Database provides the following shared pool advisory views:

- `V$SHARED_POOL_ADVICE`
- `V$LIBRARY_CACHE_MEMORY`
- `V$JAVA_POOL_ADVICE`
- `V$JAVA_LIBRARY_CACHE_MEMORY`

These shared pool advisory views provide information about library cache memory, enabling you to predict how changing the size of the shared pool can affect aging out of objects in the shared pool. The shared pool advisory statistics in these views track the library cache's use of shared pool memory and predict how the library cache will behave in shared pools of different sizes. Using these views enable you to determine:

- How much memory the library cache is using
- How much memory is currently pinned
- How much memory is on the shared pool's Least Recently Used (LRU) list
- How much time might be lost or gained by changing the size of the shared pool

These views display shared pool advisory statistics when the shared pool advisory is enabled. The statistics reset when the advisory is disabled.

The following sections describe these views in more detail:

- About the V$SHARED_POOL_ADVICE View
- About the V$LIBRARY_CACHE_MEMORY View
- About V$JAVA_POOL_ADVICE and V$JAVA_LIBRARY_CACHE_MEMORY Views

About the V$SHARED_POOL_ADVICE View

The `V$SHARED_POOL_ADVICE` view displays information about estimated parse time in the shared pool for different pool sizes. The sizes range from 10% of the current shared pool size or the amount of pinned library cache memory—whichever is higher—to 200% of the current shared pool size, in equal intervals. The value of the interval depends on the current size of the shared pool.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `V$SHARED_POOL_ADVICE` view

About the V$LIBRARY_CACHE_MEMORY View

The `V$LIBRARY_CACHE_MEMORY` view displays information about memory allocated to library cache memory objects in different namespaces. A memory object is an internal grouping of memory for efficient management. A library cache object may consist of one or more memory objects.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `V$LIBRARY_CACHE_MEMORY` view

About V$JAVA_POOL_ADVICE and V$JAVA_LIBRARY_CACHE_MEMORY Views

The `V$JAVA_POOL_ADVICE` and `V$JAVA_LIBRARY_CACHE_MEMORY` views contain Java pool advisory statistics that track information about library cache memory used for Java and predict how changing the size of the Java pool affects the parse rate.

The `V$JAVA_POOL_ADVICE` view displays information about estimated parse time in the Java pool for different pool sizes. The sizes range from 10% of the current Java pool size or the amount of pinned Java library cache memory—whichever is higher—to 200% of the current Java pool size, in equal intervals. The value of the interval depends on the current size of the Java pool.

> **See Also:**
>
> *Oracle Database Reference* for more information about the "V$JAVA_POOL_ADVICE" and "V$JAVA_LIBRARY_CACHE_MEMORY" views

Using Dictionary Cache Statistics

Typically, if the shared pool is adequately sized for the library cache, it will also be adequate sized for the dictionary cache data.

Misses on the data dictionary cache are to be expected in some cases. When the database instance starts up, the data dictionary cache does not contain any data. Therefore, any SQL statement issued is likely to result in cache misses. As more data is read into the cache, the likelihood of cache misses decreases. Eventually, the database reaches a steady state, in which the most frequently-used dictionary data is in the cache. At this point, very few cache misses occur.

Each row in the V$ROWCACHE view contains statistics for a single type of data dictionary item. These statistics reflect all data dictionary activity since the most recent instance startup.

Table 14-1 lists the columns in the V$ROWCACHE view that reflect the use and effectiveness of the data dictionary cache.

Table 14-1 Data Dictionary Columns in the V$ROWCACHE View

Column	Description
PARAMETER	Identifies a particular data dictionary item. For each row, the value in this column is the item prefixed by dc_. For example, in the row that contains statistics for file descriptions, this column contains the value dc_files.
GETS	Shows the total number of requests for information about the corresponding item. For example, in the row that contains statistics for file descriptions, this column contains the total number of requests for file description data.
GETMISSES	Shows the number of data requests that are not satisfied by the cache and required an I/O.
MODIFICATIONS	Shows the number of times data in the dictionary cache was updated.

Example 14-3 shows a query of this view to monitor the statistics over a period while the application is running. The derived column PCT_SUCC_GETS can be considered as the item-specific hit ratio.

Example 14-3 Querying the V$ROWCACHE View

```
column parameter format a21
column pct_succ_gets format 999.9
column updates format 999,999,999

SELECT parameter,
       sum(gets),
       sum(getmisses),
       100*sum(gets - getmisses) / sum(gets) pct_succ_gets,
       sum(modifications) updates
  FROM V$ROWCACHE
 WHERE gets > 0
 GROUP BY parameter;
```

The output of this query might look like the following:

```
PARAMETER             SUM(GETS) SUM(GETMISSES) PCT_SUCC_GETS      UPDATES
--------------------- ---------- -------------- ------------- ------------
dc_database_links             81              1          98.8            0
dc_free_extents            44876          20301          54.8       40,453
dc_global_oids                42              9          78.6            0
dc_histogram_defs           9419            651          93.1            0
dc_object_ids              29854            239          99.2           52
dc_objects                 33600            590          98.2           53
dc_profiles                19001              1         100.0            0
dc_rollback_segments       47244             16         100.0           19
dc_segments               100467          19042          81.0       40,272
dc_sequence_grants           119             16          86.6            0
dc_sequences               26973             16          99.9       26,811
```

```
dc_synonyms                 6617          168         97.5            0
dc_tablespace_quotas         120            7         94.2           51
dc_tablespaces            581248           10        100.0            0
dc_used_extents            51418        20249         60.6       42,811
dc_user_grants             76082           18        100.0            0
dc_usernames              216860           12        100.0            0
dc_users                  376895           22        100.0            0
```

In this example, the output shows the following:

- There are large numbers of misses and updates for used extents, free extents, and segments. This implies that the database instance had a significant amount of dynamic space extension.
- Comparing the percentage of successful gets with the actual number of gets indicates the shared pool is large enough to adequately store dictionary cache data.

You can also calculate the overall dictionary cache hit ratio using the following query; however, summing up the data over all the caches will lose the finer granularity of data:

```
SELECT (SUM(gets - getmisses - fixed)) / SUM(gets) "row cache"
  FROM V$ROWCACHE;
```

Increasing Memory Allocated to the Shared Pool

Increasing the amount of memory for the shared pool increases the amount of memory available to the library cache, the dictionary cache, and the result cache. Before doing so, review the shared pool statistics and examine:

- If the value of the `V$LIBRARYCACHE.RELOADS` column is near zero
- If the ratio of total `V$ROWCACHE.GETMISSES` column to total `V$ROWCACHE.GETS` is less than 10% or 15% for frequently-accessed dictionary caches, depending on the application

If both of these conditions are met, then the shared pool is adequately sized and increasing its memory will likely not improve performance. On the other hand, if either of these conditions is not met, and the application is using the shared pool effectively, as described in "Using the Shared Pool", then consider increasing the memory of the shared pool.

To increase the size of the shared pool:

- Increase the value of the `SHARED_POOL_SIZE` initialization parameter until the conditions are met.

  The maximum value for this parameter depends on your operating system. This measure reduces implicit reparsing of SQL statements and PL/SQL blocks on execution.

IC - Need link to "Managing Server Result Cache Memory with Init Parameters"

Reducing Memory Allocated to the Shared Pool

If the value of the `V$LIBRARYCACHE.RELOADS` column is near zero, and there is a small amount of free memory in the shared pool, then the shared pool is adequately sized to store the most frequently-accessed data. If there are always significant amounts of free memory in the shared pool and you want to allocate this memory elsewhere, then consider reducing the shared pool size.

To decrease the size of the shared pool

- Reduce the value of the `SHARED_POOL_SIZE` initialization parameter, while ensuring that good performance is maintained.

Deallocating Cursors

If there are no library cache misses, then consider setting the value of the `CURSOR_SPACE_FOR_TIME` initialization parameter to `TRUE` to accelerate execution calls. This parameter specifies whether a cursor can be deallocated from the library cache to make room for a new SQL statement.

If the `CURSOR_SPACE_FOR_TIME` parameter is set to:

- `FALSE` (the default), then a cursor can be deallocated from the library cache regardless of whether application cursors associated with its SQL statement are open.

  In this case, Oracle Database must verify that the cursor containing the SQL statement is in the library cache.

- `TRUE`, then a cursor can be deallocated only when all application cursors associated with its statement are closed.

  In this case, Oracle Database does not need to verify that a cursor is in the library cache because it cannot be deallocated while an application cursor associated with it is open.

Setting the value of the parameter to `TRUE` saves Oracle Database a small amount of time and may slightly improve the performance of execution calls. This value also prevents the deallocation of cursors until associated application cursors are closed.

Do not set the value of the `CURSOR_SPACE_FOR_TIME` parameter to `TRUE` if:

- Library cache misses are found in execution calls

  Library cache misses indicate that the shared pool is not large enough to hold the shared SQL areas of all concurrently open cursors. If the shared pool does not have enough space for a new SQL statement and the value for this parameter is set to `TRUE`, then the statement cannot be parsed and Oracle Database returns an error indicating that there is not enough shared memory.

- The amount of memory available to each user for private SQL areas is scarce

  This value also prevents the deallocation of private SQL areas associated with open cursors. If the private SQL areas for all concurrently open cursors fills the available memory so that there is no space for a new SQL statement, then the statement cannot be parsed and Oracle Database returns an error indicating that there is not enough memory.

If the shared pool does not have enough space for a new SQL statement and the value of this parameter is set to `FALSE`, then Oracle Database deallocates an existing cursor. Although deallocating a cursor may result in a subsequent library cache miss (if the cursor is re-executed), this is preferable to an error halting the application because a SQL statement cannot be parsed.

Caching Session Cursors

The session cursor cache contains closed session cursors for SQL and PL/SQL, including recursive SQL. This cache can be useful to applications that use Oracle Forms because switching from one form to another closes all session cursors associated with the first form. If an application repeatedly issues parse calls on the same set of SQL statements, then reopening session cursors can degrade performance. By reusing cursors, the database reduces parse times, leading to faster overall execution times.

This section describes the session cursor cache and contains the following topics:

About the Session Cursor Cache

A session cursor represents an instantiation of a shared child cursor, which is stored in the shared pool, for a specific session. Each session cursor stores a reference to a child cursor that it has instantiated.

Oracle Database checks the library cache to determine whether more than three parse requests have been issued on a given statement. If a cursor has been closed three times, then Oracle Database assumes that the session cursor associated with the statement should be cached and moves the cursor into the session cursor cache.

Subsequent requests to parse a SQL statement by the same session search an array for pointers to the shared cursor. If the pointer is found, then the database dereferences the pointer to determine whether the shared cursor exists. To reuse a cursor from the cache, the cache manager checks whether the cached states of the cursor match the current session and system environment.

> **Note:**
>
> Reuse of a cached cursor still registers as a parse, even though it is not a hard parse.

An LRU algorithm removes entries in the session cursor cache to make room for new entries when needed. The cache also uses an internal time-based algorithm to age out cursors that have been idle for an certain amount of time.

Enabling the Session Cursor Cache

The following initialization parameters pertain to the session cursor cache:

- `SESSION_CACHED_CURSORS`

  This parameter sets the maximum number of cached closed cursors for each session. The default value is 50. Use this parameter to reuse cursors from the cache for the statements that get executed repeatedly in the same session.

- `OPEN_CURSORS`

This parameter specifies the maximum number of cursors a session can have open simultaneously. For example, if its value is set to 1000, then each session can have up to 1000 cursors open at one time.

These parameters are independent. For example, you can set the value of the `SESSION_CACHED_CURSORS` parameter higher than the value of the `OPEN_CURSORS` parameter because session cursors are not cached in an open state.

To enable the session cursor cache:

1. Determine the maximum number of session cursors to keep in the cache.
2. Do one of the following:
   - To enable static caching, set the value of the `SESSION_CACHED_CURSORS` parameter to the number determined in the previous step.
   - To enable dynamic caching, execute the following statement:

     ```
     ALTER SESSION SET SESSION_CACHED_CURSORS = value;
     ```

Sizing the Session Cursor Cache

Use the `V$SESSTAT` view to determine if the session cursor cache is adequately sized for the database instance.

To size the session cursor cache:

1. Query the `V$SESSTAT` view to determine how many cursors are currently cached in a particular session.
2. Query the `V$SESSTAT` view to find the percentage of parse calls that found a cursor in the session cursor cache.
3. Consider increasing the value of the `SESSION_CACHED_CURSORS` parameter if the following conditions are true:
   - The session cursor cache count is close to the maximum
   - The percentage of session cursor cache hits is low relative to the total parses
   - The application repeatedly performs parse calls for the same queries

Example 14-4 shows two queries of this view.

Example 14-4 Querying the V$SESSTAT View

The following query finds how many cursors are currently cached in a particular session:

```
SELECT a.value curr_cached, p.value max_cached,
       s.username, s.sid, s.serial#
  FROM v$sesstat a, v$statname b, v$session s, v$parameter2 p
 WHERE a.statistic# = b.statistic#  and s.sid=a.sid and a.sid=&sid
   AND p.name='session_cached_cursors'
   AND b.name = 'session cursor cache count';
```

The output of this query might look like the following:

```
CURR_CACHED MAX_CACHED USERNAME   SID    SERIAL#
----------- ---------- -------- ----- ----------
         49 50         APP         35        263
```

This output shows that the number of cursors currently cached for session 35 is close to the maximum.

The following query finds the percentage of parse calls that found a cursor in the session cursor cache:

```
SELECT cach.value cache_hits, prs.value all_parses,
       round((cach.value/prs.value)*100,2) as "% found in cache"
  FROM v$sesstat cach, v$sesstat prs, v$statname nm1, v$statname nm2
 WHERE cach.statistic# = nm1.statistic#
   AND nm1.name = 'session cursor cache hits'
   AND prs.statistic#=nm2.statistic#
   AND nm2.name= 'parse count (total)'
   AND cach.sid= &sid and prs.sid= cach.sid;
```

The output of this query might look like the following:

```
CACHE_HITS ALL_PARSES % found in cache
---------- ---------- ---------------
        34        700            4.57
```

This output shows that the number of hits in the session cursor cache for session 35 is low compared to the total number of parses.

In this example, setting the value of the `SESSION_CACHED_CURSORS` parameter to 100 may help boost performance.

Sharing Cursors

In the context of SQL parsing, an identical statement is a SQL statement whose text is identical to another statement, character for character, including spaces, case, and comments. A similar statement is identical except for the values of some literals.

The parse phase compares the statement text with statements in the shared pool to determine if the statement can be shared. If the value of the `CURSOR_SHARING` initialization parameter is set to `EXACT` (the default value), and if a statement in the shared pool is not identical, then the database does not share the SQL area. Instead, each SQL statement has its own parent cursor and its own execution plan based on the literal in the statement.

This section describes how cursors can be shared and contains the following topics:

- About Cursor Sharing
- Forcing Cursor Sharing

About Cursor Sharing

When SQL statements use literals rather than bind variables, setting the value of the `CURSOR_SHARING` initialization parameter to `FORCE` enables the database to replace literals with system-generated bind variables. Using this technique, the database may reduce the number of parent cursors in the shared SQL area.

When the value of the `CURSOR_SHARING` parameter is set to `FORCE`, the database performs the following steps during the parse phase:

1. Searches for an identical statement in the shared pool.

   If an identical statement is found, then the database skips the next step and proceeds to step 3. Otherwise, the database proceeds to the next step.

2. Searches for a similar statement in the shared pool.

   If a similar statement is *not* found, then the database performs a hard parse. If a similar statement *is* found, then the database proceeds to the next step.

3. Proceeds through the remaining steps of the parse phase to ensure that the execution plan of the existing statement is applicable to the new statement.

   If the plan is not applicable, then the database performs a hard parse. If the plan is applicable, then the database proceeds to the next step.

4. Shares the SQL area of the statement.

For details about the various checks performed by the database, see "SQL Sharing Criteria".

Forcing Cursor Sharing

The best practice is to write sharable SQL and use the default value of `EXACT` for the `CURSOR_SHARING` initialization parameter. By default, Oracle Database uses adaptive cursor sharing to enable a single SQL statement that contains bind variables to use multiple execution plans. However, for applications with many similar statements that use literals instead of bind variables, setting the value of the `CURSOR_SHARING` parameter to `FORCE` may improve cursor sharing, resulting in reduced memory usage, faster parses, and reduced latch contention. Consider this approach when statements in the shared pool differ only in the values of literals, and when response time is poor because of a high number of library cache misses. In this case, setting the value of the `CURSOR_SHARING` parameter to `FORCE` maximizes cursor sharing and leverages adaptive cursor sharing to generate multiple execution plans based on different literal value ranges.

If stored outlines are generated with the value of the `CURSOR_SHARING` parameter set to `EXACT`, then the database does not use stored outlines generated with literals. To avoid this problem, generate outlines with `CURSOR_SHARING` set to `FORCE` and use the `CREATE_STORED_OUTLINES` parameter.

Setting the value of the `CURSOR_SHARING` parameter to `FORCE` has the following drawbacks:

- The database must perform extra work during the soft parse to find a similar statement in the shared pool.
- There is an increase in the maximum lengths (as returned by `DESCRIBE`) of any selected expressions that contain literals in a `SELECT` statement. However, the actual length of the data returned does not change.
- Star transformation is not supported.

When the value of the `CURSOR_SHARING` parameter is set to `FORCE`, the database uses one parent cursor and one child cursor for each distinct SQL statement. The same plan is used for each execution of the same statement. For example, consider the following SQL statement:

```
SELECT *
  FROM hr.employees
 WHERE employee_id = 101;
```

If the value of the `CURSOR_SHARING` parameter is set to `FORCE`, then the database optimizes this statement as if it contained a bind variable and uses bind peeking to estimate cardinality.

> **Note:**
>
> Starting with Oracle Database 11*g* Release 2, setting the value of the `CURSOR_SHARING` parameter to `SIMILAR` is obsolete. Consider using adaptive cursor sharing instead.

> **See Also:**
>
> - *Oracle Database Reference* for information about the `CURSOR_SHARING` initialization parameter
> - *Oracle Database SQL Tuning Guide* for information about adaptive cursor sharing

Keeping Large Objects to Prevent Aging

After an entry is loaded into the shared pool, it cannot be moved. Sometimes, as entries are loaded and aged out, the free memory may become fragmented. Shared SQL and PL/SQL areas age out of the shared pool according to an LRU algorithm that is similar to database buffers. To improve performance and avoid reparsing, prevent large SQL or PL/SQL areas from aging out of the shared pool.

The `DBMS_SHARED_POOL` package enables you to keep objects in shared memory, so that they do not age out with the normal LRU mechanism. By using the `DBMS_SHARED_POOL` package to load the SQL and PL/SQL areas before memory fragmentation occurs, the objects can be kept in memory. This ensures that memory is available and prevents the sudden slowdowns in user response times that occur when SQL and PL/SQL areas are accessed after being aged out.

Consider using the `DBMS_SHARED_POOL` package:

- When loading large PL/SQL objects, such as the `STANDARD` and `DIUTIL` packages.

  When large PL/SQL objects are loaded, user response time may be affected if smaller objects must age out of the shared pool to make room for the larger objects. In some cases, there may be insufficient memory to load the large objects.
- To keep compiled triggers on frequently used tables in the shared pool.
- Support sequences.

  Sequence numbers are lost when a sequence ages out of the shared pool. The `DBMS_SHARED_POOL` package keeps sequences in the shared pool, thus preventing the loss of sequence numbers.

To keep a SQL or PL/SQL area in shared memory:

1. Decide which packages or cursors to keep in memory.
2. Start up the database.
3. Call the `DBMS_SHARED_POOL.KEEP` package to pin the objects.

This procedure ensures that the system does not run out of shared memory before the pinned objects are loaded. Pinning the objects early in the life of the database instance prevents memory fragmentation that may result from keeping a large portion of memory in the middle of the shared pool.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about using `DBMS_SHARED_POOL` procedures

Configuring the Reserved Pool

Although Oracle Database breaks down very large requests for memory into smaller chunks, on some systems there may be a requirement to find a contiguous chunk of memory (such as over 5 KB, the default minimum reserved pool allocation is 4,400 bytes).

If there is not enough free space in the shared pool, then Oracle Database must search for and free enough memory to satisfy this request. This operation may hold the latch resource for significant periods of time, causing minor disruption to other concurrent attempts at memory allocation.

To avoid this, Oracle Database internally reserves a small memory area in the shared pool by default that the database can use if the shared pool does not have enough space. This reserved pool makes allocation of large chunks more efficient. The database can use this memory for operations such as PL/SQL and trigger compilation, or for temporary space while loading Java objects. After the memory allocated from the reserved pool is freed, it is returned to the reserved pool.

For large allocations, Oracle Database attempts to allocate space in the shared pool in the following order:

1. From the unreserved part of the shared pool.
2. From the reserved pool.

   If there is not enough space in the unreserved part of the shared pool, then Oracle Database checks whether the reserved pool has enough space.
3. From memory.

   If there is not enough space in the unreserved and reserved parts of the shared pool, then Oracle Database attempts to free enough memory for the allocation. The database then retries the unreserved and reserved parts of the shared pool.

This section describes how to configure the reserved pool and contains the following topics:

- Sizing the Reserved Pool
- Increasing Memory Allocated to the Reserved Pool
- Reducing Memory Allocated to the Reserved Pool

Sizing the Reserved Pool

Typically, it is not necessary to change the default amount of space Oracle Database reserves for the reserved pool. However, there may be cases where you need to set aside space in the shared pool for unusually large allocations of memory.

You can set the reserved pool size by setting the value of the `SHARED_POOL_RESERVED_SIZE` initialization parameter. The default value for the `SHARED_POOL_RESERVED_SIZE` parameter is 5% of the `SHARED_POOL_SIZE` parameter.

If you set the value of the `SHARED_POOL_RESERVED_SIZE` parameter to more than half of the `SHARED_POOL_SIZE` parameter, then Oracle Database returns an error because the database does not allow you to reserve too much memory for the reserved pool. The amount of operating system memory available may also constrain the size of the shared pool. In general, set the value of the `SHARED_POOL_RESERVED_SIZE` parameter to no higher than 10% of the `SHARED_POOL_SIZE` parameter. On most systems, this value is sufficient if the shared pool is adequately tuned. If you increase this value, then the database takes additional memory from the shared pool and reduces the amount of unreserved shared pool memory available for smaller allocations.

When tuning these parameters, use statistics from the `V$SHARED_POOL_RESERVED` view. On a system with ample free memory to increase the size of the SGA, the value of the `REQUEST_MISSES` statistic should equal zero. If the system is constrained by operating system memory, then the goal is to have the `REQUEST_FAILURES` statistic equal zero, or at least prevent its value from increasing. If you cannot achieve these target values, then increase the value of the `SHARED_POOL_RESERVED_SIZE` parameter. Also, increase the value of the `SHARED_POOL_SIZE` parameter by the same amount, because the reserved list is taken from the shared pool.

The `V$SHARED_POOL_RESERVED` fixed view can also indicate when the value of the `SHARED_POOL_SIZE` parameter is too small. This can be the case if the `REQUEST_FAILURES` statistic is greater than zero and increasing. If the reserved list is enabled, then decrease the value of the `SHARED_POOL_RESERVED_SIZE` parameter. If the reserved list is not enabled, then increase the value of the `SHARED_POOL_SIZE` parameter, as described in "Increasing Memory Allocated to the Shared Pool".

Increasing Memory Allocated to the Reserved Pool

The reserved pool is too small if the value of the `REQUEST_FAILURES` statistic is higher than zero and increasing. In this case, increase the amount of memory available to the reserved pool.

> **Note:**
>
> Increasing the amount of memory available on the reserved list does not affect users who do not allocate memory from the reserved list.

To increase the size of the reserved pool:

- Increase the value of the `SHARED_POOL_RESERVED_SIZE` and `SHARED_POOL_SIZE` initialization parameters accordingly.

The values that you select for these parameters depend on the system's SGA size constraints, as described in "Sizing the Reserved Pool".

Reducing Memory Allocated to the Reserved Pool

The reserved pool is too large if the:

- `REQUEST_MISSES` statistic is zero or not increasing
- `FREE_SPACE` statistic is greater than or equal to 50% of the `SHARED_POOL_RESERVED_SIZE` minimum

If either of these conditions is true, then reduce the amount of memory available to the reserved pool.

To reduce the size of the reserved pool:

- Decrease the value of the `SHARED_POOL_RESERVED_SIZE` initialization parameter.

Configuring the Large Pool

Unlike the shared pool, the large pool does not have an LRU list. Oracle Database does not attempt to age objects out of the large pool. Consider configuring a large pool if the database instance uses any of the following Oracle Database features:

- Shared server

  In a shared server architecture, the session memory for each client process is included in the shared pool.

- Parallel query

  Parallel query uses shared pool memory to cache parallel execution message buffers.

- Recovery Manager

  Recovery Manager (RMAN) uses the shared pool to cache I/O buffers during backup and restore operations. For I/O server processes, backup, and restore operations, Oracle Database allocates buffers that are a few hundred kilobytes in size.

This section describes how to configure the large pool for the shared server architecture and contains the following topics:

- Configuring the Large Pool for Shared Server Architecture
- Configuring the Large Pool for Parallel Query
- Sizing the Large Pool
- Limiting Memory Use for User Sessions
- Reducing Memory Use Using Three-Tier Connections

See Also:

- *Oracle Database Concepts* for information about the large pool
- *Oracle Database Backup and Recovery User's Guide* for information about sizing the large pool with Recovery Manager

Configuring the Large Pool for Shared Server Architecture

As Oracle Database allocates shared pool memory to shared server sessions, the amount of shared pool memory available for the library cache and data dictionary cache decreases. If you allocate the shared server session memory from a different pool, then the shared pool can be reserved for caching shared SQL.

Oracle recommends using the large pool to allocate the User Global Area (UGA) for the shared server architecture. Using the large pool instead of the shared pool decreases fragmentation of the shared pool and eliminates the performance overhead from shrinking the shared SQL cache.

By default, the large pool is not configured. If you do not configure the large pool, then Oracle Database uses the shared pool for shared server user session memory. If you do configure the large pool, Oracle Database still allocates a fixed amount of memory (about 10K) for each configured session from the shared pool when a shared server architecture is used. In either case, consider increasing the size of the shared pool accordingly.

Note:

Even though use of shared memory increases with shared servers, the total amount of memory use decreases. This is because there are fewer processes; therefore, Oracle Database uses less PGA memory with shared servers when compared to dedicated server environments.

Tip:

To specify the maximum number of concurrent shared server sessions that the database allows, use the `CIRCUITS` initialization parameter.

Tip:

For best performance with sort operations using shared servers, set the values of the `SORT_AREA_SIZE` and `SORT_AREA_RETAINED_SIZE` initialization parameters to the same value. This keeps the sort result in the large pool instead of writing it to disk.

Configuring the Large Pool for Parallel Query

Parallel query uses shared pool memory to cache parallel execution message buffers when Automatic Memory Management or Automatic Shared Memory Management is not enabled. Caching parallel execution message buffers in the shared pool increases its workload and may cause fragmentation.

To avoid possible negative impact to performance, Oracle recommends that you do not manage SGA memory manually when parallel query is used. Instead, you should enable Automatic Memory Management or Automatic Shared Memory Management to ensure that the large pool will be used to cache parallel execution memory buffers.

See Also:

- "Automatic Memory Management"
- "Automatic Shared Memory Management"
- *Oracle Database VLDB and Partitioning Guide*

Sizing the Large Pool

When storing shared server-related UGA in the large pool, the exact amount of UGA that Oracle Database uses depends on the application. Each application requires a different amount of memory for session information, and configuration of the large pool should reflect the memory requirement.

Oracle Database collects statistics on memory used by a session and stores them in the `V$SESSTAT` view. Table 14-2 lists the statistics from this view that reflect session UGA memory.

Table 14-2 Memory Statistics in the V$SESSTAT View

Statistic	Description
`session UGA memory`	Shows the amount of memory in bytes allocated to the session.
`session UGA memory max`	Shows the maximum amount of memory in bytes ever allocated to the session.

There are two methods to use this view to determine a correct size for the large pool. One method is to configure the size of the large pool based on the number of simultaneously active sessions. To do this, observe UGA memory usage for a typical user and multiply this amount by the estimated number of user sessions. For example, if the shared server requires 200K to 300K to store session information for a typical user session and you anticipate 100 active user sessions simultaneously, then configure the large pool to 30 MB.

Another method is to calculate the total and maximum memory being used by all user sessions. Example 14-5 shows two queries of the `V$SESSTAT` and `V$STATNAME` views to do this.

Example 14-5 Querying the V$SESSTAT and V$STATNAME Views

While the application is running, issue the following queries:

```
SELECT SUM(value) || ' bytes' "total memory for all sessions"
  FROM V$SESSTAT, V$STATNAME
 WHERE name = 'session uga memory'
   AND V$SESSTAT.STATISTIC# = V$STATNAME.STATISTIC#;

SELECT SUM(value) || ' bytes' "total max mem for all sessions"
  FROM V$SESSTAT, V$STATNAME
 WHERE name = 'session uga memory max'
   AND V$SESSTAT.STATISTIC# = V$STATNAME.STATISTIC#;
```

These queries also select from the `V$STATNAME` view to obtain internal identifiers for `session memory` and `max session memory`.

The output of these queries might look like the following:

```
TOTAL MEMORY FOR ALL SESSIONS
-----------------------------
157125 BYTES

TOTAL MAX MEM FOR ALL SESSIONS
------------------------------
417381 BYTES
```

The result of the first query shows that the memory currently allocated to all sessions is 157,125 bytes. This value is the total memory with a location that depends on how the sessions are connected to the database. If the sessions are connected to dedicated server processes, then this memory is part of the memories of the user processes. If the sessions are connected to shared server processes, then this memory is part of the shared pool.

The result of the second query shows that the sum of the maximum size of the memory for all sessions is 417,381 bytes. The second result is greater than the first because some sessions have deallocated memory since allocating their maximum amounts.

Use the result of either queries to determine the correct size for the shared pool. The first value is likely to be a better estimate than the second, unless if you expect all sessions to reach their maximum allocations simultaneously.

To size the large pool:

1. Verify the pool (shared pool or large pool) in which the memory for an object resides by checking the `POOL` column in the `V$SGASTAT` view.
2. Set a value for the `LARGE_POOL_SIZE` initialization parameter.

   The minimum value for this parameter is 300K.

Limiting Memory Use for User Sessions

To restrict the memory used by each client session from the SGA, set a resource limit using `PRIVATE_SGA`.

`PRIVATE_SGA` defines the number of bytes of memory used from the SGA by a session. However, this parameter is rarely used, because most DBAs do not limit SGA consumption on a user-by-user basis.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about setting the `PRIVATE_SGA` resource limit

Reducing Memory Use Using Three-Tier Connections

If there is a high number of connected users, then consider reducing memory usage by implementing three-tier connections. Using a transaction process (TP) monitor is feasible only with pure transactional models because locks and uncommitted DML operations cannot be held between calls.

Using a shared server environment:

- Is much less restrictive of the application design than a TP monitor.
- Dramatically reduces operating system process count and context switches by enabling users to share a pool of servers.
- Substantially reduces overall memory usage, even though more SGA is used in shared server mode.

15

Tuning the Result Cache

This chapter describes how to tune the result cache and contains the following topics:

- About the Result Cache
- Configuring the Result Cache
- Specifying Queries for Result Caching
- Monitoring the Result Cache

About the Result Cache

A result cache is an area of memory, either in the Shared Global Area (SGA) or client application memory, that stores the results of a database query or query block for reuse. The cached rows are shared across SQL statements and sessions unless they become stale.

This section describes the two types of result cache and contains the following topics:

- Server Result Cache Concepts
- Client Result Cache Concepts

Server Result Cache Concepts

The server result cache is a memory pool within the shared pool. This memory pool consists of the SQL query result cache—which stores results of SQL queries—and the PL/SQL function result cache, which stores values returned by PL/SQL functions.

This section describes the server result cache and contains the following topics:

- Benefits of Using the Server Result Cache
- Understanding How the Server Result Cache Works

> **See Also:**
>
> - *Oracle Database Concepts* for information about the server result cache
> - *Oracle Database PL/SQL Language Reference* for information about the PL/SQL function result cache

Benefits of Using the Server Result Cache

The benefits of using the server result cache depend on the application. OLAP applications can benefit significantly from its use. Good candidates for caching are queries that access a high number of rows but return a small number, such as those in a data warehouse. For

example, you can use advanced query rewrite with equivalences to create materialized views that materialize queries in the result cache instead of using tables.

> **See Also:**
>
> *Oracle Database Data Warehousing Guide* for information about using the result cache and advance query rewrite with equivalences

Understanding How the Server Result Cache Works

When a query executes, the database searches the cache memory to determine whether the result exists in the result cache. If the result exists, then the database retrieves the result from memory instead of executing the query. If the result is not cached, then the database executes the query, returns the result as output, and stores the result in the result cache.

When users execute queries and functions repeatedly, the database retrieves rows from the cache, decreasing response time. Cached results become invalid when data in dependent database objects is modified.

The following sections contains examples of how to retrieve results from the server result cache:

- How Results are Retrieved in a Query
- How Results are Retrieved in a View

How Results are Retrieved in a Query

The following example shows a query of `hr.employees` that uses the `RESULT_CACHE` hint to retrieve rows from the server result cache.

```
SELECT /*+ RESULT_CACHE */ department_id, AVG(salary)
  FROM hr.employees
 GROUP BY department_id;
```

A portion of the execution plan of this query might look like the following:

```
------------------------------------------------------------
| Id | Operation          | Name                       |Rows
------------------------------------------------------------
| 0 | SELECT STATEMENT    |                            | 11
| 1 |  RESULT CACHE       | 8fpza04gtwsfr6n595au15yj4y |
| 2 |   HASH GROUP BY     |                            | 11
| 3 |    TABLE ACCESS FULL| EMPLOYEES                  | 107
------------------------------------------------------------
```

In this example, the results are retrieved directly from the cache, as indicated in step 1 of the execution plan. The value in the `Name` column is the cache ID of the result.

The following example shows a query of the `V$RESULT_CACHE_OBJECTS` view to retrieve detailed statistics about the cached result.

```
SELECT id, type, creation_timestamp, block_count,
       column_count, pin_count, row_count
```

```
  FROM V$RESULT_CACHE_OBJECTS
 WHERE cache_id = '8fpza04gtwsfr6n595au15yj4y';
```

In this example, the value of `CACHE_ID` is the cache ID obtained from the explain plan in the earlier example. The output of this query might look like the following:

```
        ID TYPE       CREATION_ BLOCK_COUNT COLUMN_COUNT  PIN_COUNT  ROW_COUNT
---------- ---------- --------- ----------- ------------ ---------- ----------
         2 Result     06-NOV-11           1            2          0         12
```

How Results are Retrieved in a View

Example 15-1 shows a query that uses the `RESULT_CACHE` hint within a `WITH` clause view.

Example 15-1 RESULT_CACHE Hint Specified in a WITH View

```
WITH summary AS
( SELECT /*+ RESULT_CACHE */ department_id, avg(salary) avg_sal
    FROM hr.employees
   GROUP BY department_id )
SELECT d.*, avg_sal
  FROM hr.departments d, summary s
 WHERE d.department_id = s.department_id;
```

A portion of the execution plan of this query might look like the following:

```
-----------------------------------------------------------------------------------------------
| Id| Operation             | Name                       | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|  0| SELECT STATEMENT      |                            |   11 |   517 |      7  (29)| 00:00:01 |
|* 1|  HASH JOIN            |                            |   11 |   517 |      7  (29)| 00:00:01 |
|  2|   VIEW                |                            |   11 |   286 |      4  (25)| 00:00:01 |
|  3|    RESULT CACHE       | 8nknkh64ctmz94a5muf2tyb8r  |      |       |             |          |
|  4|     HASH GROUP BY     |                            |   11 |    77 |      4  (25)| 00:00:01 |
|  5|      TABLE ACCESS FULL| EMPLOYEES                  |  107 |   749 |      3   (0)| 00:00:01 |
|  6|   TABLE ACCESS FULL   | DEPARTMENTS                |   27 |   567 |      2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

In this example, the `summary` view results are retrieved directly from the cache, as indicated in step 3 of the execution plan. The value in the `Name` column is the cache ID of the result.

Client Result Cache Concepts

The Oracle Call Interface (OCI) client result cache is a memory area inside a client process that caches SQL query result sets for OCI applications. This client cache exists for each client process and is shared by all sessions inside the process. Oracle recommends client result caching for queries of read-only or read-mostly tables.

> **Note:**
>
> The client result cache is distinct from the server result cache, which resides in the SGA. When client result caching is enabled, the query result set can be cached on the client, server, or both. Client caching can be enabled even if the server result cache is disabled.

This section describes the client result cache and contains the following topics:

Benefits of Using the Client Result Cache

OCI drivers, such as OCCI, the JDBC OCI driver, and ODP.NET, support client result caching. Performance benefits of using the client result cache include:

- Reduced query response time

  When queries are executed repeatedly, the application retrieves results directly from the client cache memory, resulting in faster query response time.

- More efficient use of database resources

  The reduction in server round trips may result in substantial performance savings of server resources, such as server CPU and I/O. These resources are freed for other tasks, thereby making the server more scalable.

- Reduced memory cost

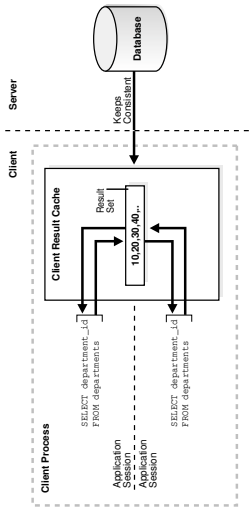
  The result cache uses client memory, which may be less expensive than server memory.

Understanding How the Client Result Cache Works

The client result cache stores the results of the outermost query, which are the columns defined by the OCI application. Subqueries and query blocks are not cached.

The following figure illustrates a client process with a database login session. This client process has one client result cache shared amongst multiple application sessions running in the client process. If the first application session runs a query, then it retrieves rows from the database and caches them in the client result cache. If other application sessions run the same query, then they also retrieve rows from the client result cache.

Figure 15-1 Client Result Cache



The client result cache transparently keeps the result set consistent with session state or database changes that affect it. When a transaction changes the data or metadata of database objects used to build the cached result, the database sends an invalidation to the OCI client on its next round trip to the server.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for details about the client result cache

Configuring the Result Cache

This section describes how to configure the server and client result cache and contains the following topics:

- Configuring the Server Result Cache
- Configuring the Client Result Cache
- Setting the Result Cache Mode
- Requirements for the Result Cache

Configuring the Server Result Cache

By default, on database startup, Oracle Database allocates memory to the server result cache in the shared pool. The memory size allocated depends on the memory size of the shared pool and the selected memory management system:

- Automatic shared memory management

  If you are managing the size of the shared pool using the `SGA_TARGET` initialization parameter, Oracle Database allocates 0.50% of the value of the `SGA_TARGET` parameter to the result cache.

- Manual shared memory management

  If you are managing the size of the shared pool using the `SHARED_POOL_SIZE` initialization parameter, then Oracle Database allocates 1% of the shared pool size to the result cache.

> **Note:**
>
> Oracle Database will not allocate more than 75% of the shared pool to the server result cache.

The size of the server result cache grows until it reaches the maximum size. Query results larger than the available space in the cache are not cached. The database employs a Least Recently Used (LRU) algorithm to age out cached results, but does not otherwise automatically release memory from the server result cache.

This section describes how to configure the server result cache and contains the following topics:

Sizing the Server Result Cache Using Initialization Parameters

Table 15-1 lists the database initialization parameters that control the server result cache.

Table 15-1 Server Result Cache Initialization Parameters

Parameter	Description
RESULT_CACHE_MAX_SIZE	Specifies the memory allocated to the server result cache. To disable the server result cache, set this parameter to 0.
RESULT_CACHE_MAX_RESULT	Specifies the maximum amount of server result cache memory (in percent) that can be used for a single result. Valid values are between 1 and 100. The default value is 5%. You can set this parameter at the system or session level.
RESULT_CACHE_REMOTE_EXPIRATION	Specifies the expiration time (in minutes) for a result in the server result cache that depends on remote database objects. The default value is 0, which specifies that results using remote objects will not be cached. If a non-zero value is set for this parameter, DML on the remote database does not invalidate the server result cache.

> **See Also:**
>
> *Oracle Database Reference* for more information about these initialization parameters

To change the memory allocated to the server result cache:

- Set the value of the RESULT_CACHE_MAX_SIZE initialization parameter to the desired size.

  In an Oracle Real Application Clusters (Oracle RAC) environment, the result cache is specific to each database instance and can be sized differently on each instance. However, invalidations work across instances. To disable the server result cache in a cluster, you must explicitly set this parameter to 0 for each instance startup.

Managing the Server Result Cache Using DBMS_RESULT_CACHE

The DBMS_RESULT_CACHE package provides statistics, information, and operators that enable you to manage memory allocation for the server result cache. Use the DBMS_RESULT_CACHE package to perform operations such as retrieving statistics on the cache memory usage and flushing the cache.

This section describes how to manage the server result cache using the DBMS_RESULT_CACHE package and contains the following topics:

- Viewing Memory Usage Statistics for the Server Result Cache
- Flushing the Server Result Cache

Viewing Memory Usage Statistics for the Server Result Cache

This section describes how to view memory allocation statistics for the result cache using the `DBMS_RESULT_CACHE` package.

To view memory usage statistics for the result cache:

- Execute the `DBMS_RESULT_CACHE.MEMORY_REPORT` procedure.

  Example 15-2 shows an execution of this procedure.

Example 15-2 Using the DBMS_RESULT_CACHE Package

```
SQL> SET SERVEROUTPUT ON
SQL> EXECUTE DBMS_RESULT_CACHE.MEMORY_REPORT
```

The output of this command might look like the following:

```
R e s u l t  C a c h e  M e m o r y  R e p o r t
[Parameters]
Block Size = 1024 bytes
Maximum Cache Size = 950272 bytes (928 blocks)
Maximum Result Size = 47104 bytes (46 blocks)
[Memory]
Total Memory = 46340 bytes [0.048% of the Shared Pool]
... Fixed Memory = 10696 bytes [0.011% of the Shared Pool]
... State Object Pool = 2852 bytes [0.003% of the Shared Pool]
... Cache Memory = 32792 bytes (32 blocks) [0.034% of the Shared Pool]
....... Unused Memory = 30 blocks
....... Used Memory = 2 blocks
........... Dependencies = 1 blocks
........... Results = 1 blocks
............... SQL = 1 blocks

PL/SQL procedure successfully completed.
```

Flushing the Server Result Cache

This section describes how to remove all existing results and purge the result cache memory using the `DBMS_RESULT_CACHE` package.

To flush the server result cache:

- Execute the `DBMS_RESULT_CACHE.FLUSH` procedure.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_RESULT_CACHE` package

Configuring the Client Result Cache

Table 15-2 lists the database initialization parameters that enable or influence the behavior of the client result cache.

Table 15-2 Client Result Cache Initialization Parameters

Parameter	Description
CLIENT_RESULT_CACHE_SIZE	Specifies the maximum size of the client result cache for each client process. To enable the client result cache, set the size to 32768 bytes or greater. A lesser value, including the default of 0, disables the client result cache. **Note:** If the CLIENT_RESULT_CACHE_SIZE setting disables the client cache, then a client node cannot enable it. If the CLIENT_RESULT_CACHE_SIZE setting enables the client cache, however, then a client node can override the setting. For example, a client node can disable client result caching or increase the size of its cache.
CLIENT_RESULT_CACHE_LAG	Specifies the amount of lag time (in milliseconds) for the client result cache. The default value is 3000 (3 seconds). If the OCI application does not perform any database calls for a period of time, then this setting forces the next statement execution call to check for validations. If the OCI application accesses the database infrequently, then setting this parameter to a low value results in more round trips from the OCI client to the database to keep the client result cache synchronized with the database.
COMPATIBLE	Specifies the release with which Oracle Database must maintain compatibility. For the client result cache to be enabled, this parameter must be set to 11.0.0.0 or higher. For client caching on views, this parameter must be set to 11.2.0.0.0 or higher.

An optional client configuration file overrides client result cache initialization parameters set in the server parameter file.

> **Note:**
>
> The client result cache lag can only be set with the CLIENT_RESULT_CACHE_LAG initialization parameter.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide* for information about the parameters that can be set in the client configuration file
> - *Oracle Database Reference* for more information about these client result cache initialization parameters

Setting the Result Cache Mode

The result cache mode is a database setting that determines which queries are eligible to store result sets in the server and client result caches. If a query is eligible for caching, then the application checks the result cache to determine whether the query result set exists in the cache. If it exists, then the result is retrieved directly from the result cache. Otherwise, the database executes the query and returns the result as output and stores it in the result cache. Oracle recommends result caching for queries of read-only or read-mostly database objects.

When the result cache is enabled, the database also caches queries that call non-deterministic PL/SQL functions. When caching `SELECT` statements that call such functions, the result cache tracks data dependencies for the PL/SQL functions and the database objects. However, if the function uses data that are not being tracked (such as sequences, `SYSDATE`, `SYS_CONTEXT`, and package variables), using the result cache on queries that call this function can produce stale results. In this regard, the behavior of the result cache is identical to caching PL/SQL functions. Therefore, always consider data accuracy, as well as performance, when choosing to enable the result cache.

To set the result cache mode:

- Set the value of the `RESULT_CACHE_MODE` initialization parameter to determine the behavior of the result cache.

  You can set this parameter for the instance (`ALTER SYSTEM`), session (`ALTER SESSION`), or in the server parameter file.

  Table 15-3 describes the values for this parameter.

Table 15-3 Values for the RESULT_CACHE_MODE Parameter

Value	Description
`MANUAL`	Query results can only be stored in the result cache by using a query hint or table annotation. This is the default and recommended value.
`FORCE`	All results are stored in the result cache. If a query result is not in the cache, then the database executes the query and stores the result in the cache. Subsequent executions of the same SQL statement, including the result cache hint, retrieve data from the cache. Sessions uses these results if possible. To exclude query results from the cache, the `/*+ NO_RESULT_CACHE */` query hint must be used. **Note:** `FORCE` mode is not recommended because the database and clients will attempt to cache all queries, which may create significant performance and latching overhead. Moreover, because queries that call non-deterministic PL/SQL functions are also cached, enabling the result cache in such a broad-based manner may cause material changes to the results.

> **See Also:**
>
> *Oracle Database Reference* for information about the `RESULT_CACHE_MODE` initialization parameter

Requirements for the Result Cache

Enabling the result cache does not *guarantee* that a specific result set will be included in the server or client result cache. In order for results to be cached, the following requirements must be met:

- Read Consistency Requirements
- Query Parameter Requirements
- Restrictions for the Result Cache

Read Consistency Requirements

For a snapshot to be reusable, it must have read consistency. For a result set to be eligible for caching, at least one of the following conditions must be true:

- The read-consistent snapshot used to build the result must retrieve the most current, committed state of the data.
- The query points to an explicit point in time using flashback query.

If the current session has an active transaction referencing objects in a query, then the results from this query are not eligible for caching.

Query Parameter Requirements

Cache results can be reused if they are parameterized with variable values when queries are equivalent and the parameter values are the same. Different values or bind variable names may cause cache misses. Results are parameterized if any of the following constructs are used in the query:

- Bind variables
- The SQL functions `DBTIMEZONE`, `SESSIONTIMEZONE`, `USERENV/SYS_CONTEXT` (with constant variables), `UID`, and `USER`
- NLS parameters

Restrictions for the Result Cache

Results cannot be cached when the following objects or functions are in a query:

- Temporary tables and tables in the `SYS` or `SYSTEM` schemas
- Sequence `CURRVAL` and `NEXTVAL` pseudo columns
- SQL functions `CURRENT_DATE`, `CURRENT_TIMESTAMP`, `LOCAL_TIMESTAMP`, `USERENV/SYS_CONTEXT` (with non-constant variables), `SYS_GUID`, `SYSDATE`, and `SYSTIMESTAMP`

The client result cache has additional restrictions for result caching.

> **Note:**
>
> Result cache does not work on an Active Data Guard standby database opened in read-only mode.

> **See Also:**
>
> *Oracle Call Interface Programmer's Guide* for information about additional restrictions for the client result cache

Specifying Queries for Result Caching

This section describes how to specify queries for result caching and contains the following topics:

- Using SQL Result Cache Hints
- Using Result Cache Table Annotations

Using SQL Result Cache Hints

Use result cache hints at the application level to control caching behavior. The SQL result cache hints take precedence over the result cache mode and result cache table annotations.

This section describes how to use SQL result cache hints and contains the following topics:

- Using the RESULT_CACHE Hint
- Using the NO_RESULT_CACHE Hint
- Using the RESULT_CACHE Hint in Views

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the `RESULT_CACHE` and `NO_RESULT_CACHE` hints

Using the RESULT_CACHE Hint

When the result cache mode is `MANUAL`, the `/*+ RESULT_CACHE */` hint instructs the database to cache the results of a query block and to use the cached results in future executions.

Example 15-3 shows a query that uses the `RESULT_CACHE` hint.

Example 15-3 Using the RESULT_CACHE Hint

```
SELECT /*+ RESULT_CACHE */ prod_id, SUM(amount_sold)
  FROM sales
 GROUP BY prod_id
 ORDER BY prod_id;
```

In this example, the query instructs the database to cache rows for a query of the `sales` table.

Using the NO_RESULT_CACHE Hint

The `/*+ NO_RESULT_CACHE */` hint instructs the database *not* to cache the results in either the server or client result caches.

Example 15-4 shows a query that uses the `NO_RESULT_CACHE` hint.

Example 15-4 Using the NO_RESULT_CACHE Hint

```
SELECT /*+ NO_RESULT_CACHE */ prod_id, SUM(amount_sold)
  FROM sales
 GROUP BY prod_id
 ORDER BY prod_id;
```

In this example, the query instructs the database not to cache rows for a query of the `sales` table.

Using the RESULT_CACHE Hint in Views

The `RESULT_CACHE` hint applies only to the query block in which the hint is specified. If the hint is specified only in a view, then only these results are cached. View caching has the following characteristics:

- The view must be one of the following types:
  - A standard view (a view created with the `CREATE ... VIEW` statement)
  - An inline view specified in the `FROM` clause of a `SELECT` statement
  - An inline view created with the `WITH` clause
- The result of a view query with a correlated column (a reference to an outer query block) cannot be cached.
- Query results are stored in the server result cache, not the client result cache.
- A caching view is not merged into its outer (or referring) query block.

  Adding the `RESULT_CACHE` hint to inline views disables optimizations between the outer query and inline view to maximize reusability of the cached result.

The following example shows a query of the inline view `view1`.

```
SELECT *
  FROM ( SELECT /*+ RESULT_CACHE */ department_id, manager_id, count(*) count
           FROM hr.employees
          GROUP BY department_id, manager_id ) view1
 WHERE department_id = 30;
```

In this example, the `SELECT` statement from `view1` is the outer block, whereas the `SELECT` statement from `employees` is the inner block. Because the `RESULT_CACHE` hint is specified only in the inner block, the results of the inner query are stored in the server result cache, but the results of the outer query are not cached.

Assume that the same session run a query of the view `view2` as shown in the following example.

```
WITH view2 AS
( SELECT /*+ RESULT_CACHE */ department_id, manager_id, count(*) count
    FROM hr.employees
   GROUP BY department_id, manager_id )
SELECT *
  FROM view2
 WHERE count BETWEEN 1 and 5;
```

In this example, because the `RESULT_CACHE` hint is specified only in the query block in the `WITH` clause, the results of the `employees` query are eligible to be cached. Because these results are cached from the query in the first example, the `SELECT` statement in the `WITH` clause in the second example can retrieve the cached rows.

Using Result Cache Table Annotations

You can also use table annotations to control result caching. Table annotations affect the entire query, not query segments. The primary benefit of using table annotations is avoiding the necessity of adding result cache hints to queries at the application level. Because a table annotation has a lower precedence than a SQL result cache hint, you can override table and session settings by using hints at the query level.

Table 15-4 describes the valid values for the `RESULT_CACHE` table annotation.

Table 15-4 Values for the RESULT_CACHE Table Annotation

Value	Description
`DEFAULT`	If at least one table in a query is set to `DEFAULT`, then result caching is *not* enabled at the table level for this query, unless if the `RESULT_CACHE_MODE` initialization parameter is set to `FORCE` or the `RESULT_CACHE` hint is specified. This is the default value.
`FORCE`	If all the tables of a query are marked as `FORCE`, then the query result is considered for caching. The table annotation `FORCE` takes precedence over the `RESULT_CACHE_MODE` parameter value of `MANUAL` set at the session level.

This section describes how to use the `RESULT_CACHE` table annotations and contains the following topics:

- Using the DEFAULT Table Annotation
- Using the FORCE Table Annotation

Using the DEFAULT Table Annotation

The `DEFAULT` table annotation prevents the database from caching results at the table level.

Example 15-5 shows a `CREATE TABLE` statement that uses the `DEFAULT` table annotation to create a table `sales` and a query of this table.

Example 15-5 Using the DEFAULT Table Annotation

```
CREATE TABLE sales (...) RESULT_CACHE (MODE DEFAULT);

SELECT prod_id, SUM(amount_sold)
  FROM sales
 GROUP BY prod_id
 ORDER BY prod_id;
```

In this example, the `sales` table is created with a table annotation that disables result caching. The example also shows a query of the `sales` table, whose results are not considered for caching because of the table annotation.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the `CREATE TABLE` statement and its syntax

Using the FORCE Table Annotation

The FORCE table annotation forces the database to cache results at the table level.

Using the `sales` table created in Example 15-5, assume that you decide to force result caching for this table, you can do so by using the `FORCE` table annotation.

Example 15-6 shows an `ALTER TABLE` statement that uses the `FORCE` table annotation on the `sales` table.

Example 15-6 Using the FORCE Table Annotation

```
ALTER TABLE sales RESULT_CACHE (MODE FORCE);

SELECT prod_id, SUM(amount_sold)
  FROM sales
 GROUP BY prod_id
HAVING prod_id=136;

SELECT /*+ NO_RESULT_CACHE */ *
  FROM sales
 ORDER BY time_id DESC;
```

This example includes two queries of the `sales` table. The first query, which is frequently used and returns few rows, is eligible for caching because of the table annotation. The second query, which is a one-time query that returns many rows, uses a hint to prevent result caching.

Monitoring the Result Cache

To view information about the server and client result caches, query the relevant database views and tables.

Table 15-5 describes the most useful views and tables for monitoring the result cache.

Table 15-5 Views and Tables with Information About the Result Cache

View/Table	Description
`V$RESULT_CACHE_STATISTICS`	Lists various server result cache settings and memory usage statistics.
`V$RESULT_CACHE_MEMORY`	Lists all the memory blocks in the server result cache and their corresponding statistics.

Table 15-5 (Cont.) Views and Tables with Information About the Result Cache

View/Table	Description
V$RESULT_CACHE_OBJECTS	Lists all the objects whose results are in the server result cache along with their attributes.
V$RESULT_CACHE_DEPENDENCY	Lists the dependency details between the results in the server result cache and dependencies among these results.
CLIENT_RESULT_CACHE_STATS$	Stores cache settings and memory usage statistics for the client result caches obtained from the OCI client processes. This statistics table contains entries for each client process that uses result caching. After the client processes terminate, the database removes their entries from this table. The client table contains information similar to V$RESULT_CACHE_STATISTICS.
DBA_TABLES, USER_TABLES, ALL_TABLES	Contains a RESULT_CACHE column that shows the result cache mode annotation for the table. If the table is not annotated, then this column shows DEFAULT. This column applies to both server and client result caches.

> **See Also:**
>
> *Oracle Database Reference* for more information about these views and tables.

The following example shows a query of the V$RESULT_CACHE_STATISTICS view to monitor server result cache statistics.

```
COLUMN name FORMAT a20
SELECT name, value
  FROM V$RESULT_CACHE_STATISTICS;
```

The output of this query might look like the following:

```
NAME                          VALUE
--------------------      ----------
Block Size (Bytes)              1024
Block Count Maximum             3136
Block Count Current               32
Result Size Maximum (Blocks)     156
Create Count Success               2
Create Count Failure               0
Find Count                         0
Invalidation Count                 0
Delete Count Invalid               0
Delete Count Valid                 0
```

The following example shows a query of the CLIENT_RESULT_CACHE_STATS$ table to monitor the client result cache statistics.

```
SELECT stat_id, SUBSTR(name,1,20), value, cache_id
  FROM CLIENT_RESULT_CACHE_STATS$
 ORDER BY cache_id, stat_id;
```

The output of this query might look like the following:

```
STAT_ID    NAME OF STATISTICS      VALUE   CACHE_ID
=======    ==================      =====   ========
    1      Block Size               256         124
    2      Block Count Max          256         124
    3      Block Count Current      128         124
    4      Hash Bucket Count       1024         124
    5      Create Count Success      10         124
    6      Create Count Failure       0         124
    7      Find Count                12         124
    8      Invalidation Count         8         124
    9      Delete Count Invalid       0         124
   10      Delete Count Valid         0         124
```

The `CLIENT_RESULT_CACHE_STATS$` table contains statistics entries for each active client process performing client result caching. Every client process has a unique cache ID.

To find the client connection information for the sessions performing client caching:

1. Obtain the session IDs from the `CLIENT_REGID` column in the `GV$SESSION_CONNECT_INFO` view that corresponds to the `CACHE_ID` column in the `CLIENT_RESULT_CACHE_STATS$` table.
2. Query the relevant columns from the `GV$SESSION_CONNECT_INFO` and `GV$SESSION` views.

For both server and client result cache statistics, a database that is optimized for result caching should show relatively low values for the `Create Count Failure` and `Delete Count Valid` statistics, while showing relatively high values for the `Find Count` statistic.

16

Tuning the Program Global Area

This chapter describes how to tune the Program Global Area (PGA). If you are using automatic memory management to manage the database memory on your system, then you do not need to manually tune the PGA as described in this chapter.

This chapter contains the following topics:

About the Program Global Area

The Program Global Area (PGA) is a private memory region that contains the data and control information for a server process. Only a server process can access the PGA. Oracle Database reads and writes information in the PGA on behalf of the server process. An example of such information is the run-time area of a cursor. Each time a cursor is executed, a new run-time area is created for that cursor in the PGA memory region of the server process executing that cursor.

> **Note:**
>
> Part of the run-time area can be located in the Shared Global Area (SGA) when using shared servers.

For complex queries (such as decision support queries), a big portion of the run-time area is dedicated to work areas allocated by memory intensive operators, including:

- Sort-based operators, such as `ORDER BY`, `GROUP BY`, `ROLLUP`, and window functions
- Hash-join
- Bitmap merge
- Bitmap create
- Write buffers used by bulk load operations

A sort operator uses a work area (the sort area) to perform the in-memory sorting of a set of rows. Similarly, a hash-join operator uses a work area (the hash area) to build a hash table from its left input.

Work Area Sizes

Oracle Database enables you to control and tune the sizes of work areas. Generally, bigger work areas can significantly improve the performance of a particular operator at the cost of higher memory consumption. The available work area sizes include:

- Optimal

  Optimal size is when the size of a work area is large enough that it can accommodate the input data and auxiliary memory structures allocated by its associated SQL operator. This is the ideal size for the work area.

- One-pass

  One-pass size is when the size of the work area is below optimal size and an extra pass is performed over part of the input data. With one-pass size, the response time is increased.

- Multi-pass

  Multi-pass size is when the size of the work area is below the one-pass threshold and multiple passes over the input data are needed. With multi-pass size, the response time is dramatically increased because the size of the work area is too small compared to the input data size.

For example, a serial sort operation that must sort 10 GB of data requires a little more than 10 GB to run as optimal size and at least 40 MB to run as one-pass size. If the work area is less than 40 MB, then the sort operation must perform several passes over the input data.

When sizing the work area, the goal is to have most work areas running with optimal size (more than 90%, or even 100% for pure OLTP systems), and only a small number of them running with one-pass size (less than 10%). Multi-pass executions should be avoided for the following reasons:

- Multi-pass executions can severely degrade performance.

  A high number of multi-pass work areas has an exponentially adverse effect on the response time of its associated SQL operator.

- Running one-pass executions does not require a large amount of memory.

  Only 22 MB is required to sort 1 GB of data in one-pass size.

Even for DSS systems running large sorts and hash-joins, the memory requirement for one-pass executions is relatively small. A system configured with a reasonable amount of PGA memory should not need to perform multiple passes over the input data.

Sizing the Program Global Area Using Automatic Memory Management

Automatic PGA memory management simplifies and improves the way PGA memory is allocated. By default, PGA memory management is enabled. In this mode, Oracle Database automatically sizes the PGA by dynamically adjusting the portion of the PGA memory dedicated to work areas, based on 20% of the SGA memory size. The minimum value is 10MB.

> **Note:**
>
> For backward compatibility, automatic PGA memory management can be disabled by setting the value of the `PGA_AGGREGATE_TARGET` initialization parameter to 0. When automatic PGA memory management is disabled, the maximum size of a work area can be sized with the associated `_AREA_SIZE` parameter, such as the `SORT_AREA_SIZE` initialization parameter.

This section describes how to size the PGA using automatic PGA memory management and contains the following topics:

- Configuring Automatic PGA Memory Management
- Setting the Initial Value for PGA_AGGREGATE_TARGET
- Monitoring Automatic PGA Memory Management
- Tuning PGA_AGGREGATE_TARGET

Configuring Automatic PGA Memory Management

When running Oracle Database in automatic PGA memory management mode, sizing of work areas for all sessions is automatic, and the `*_AREA_SIZE` parameters are ignored by all sessions running in this mode. Oracle Database automatically derives the total amount of PGA memory available to active work areas from the `PGA_AGGREGATE_TARGET` initialization parameter. The amount of PGA memory is set to the value of `PGA_AGGREGATE_TARGET` minus the amount of PGA memory allocated to other components of the system (such as PGA memory allocated by sessions). Oracle Database then assigns the resulting PGA memory to individual active work areas based on their specific memory requirements.

Oracle Database attempts to adhere to the `PGA_AGGREGATE_TARGET` value set by the DBA by dynamically controlling the amount of PGA memory allotted to work areas. To accomplish this, Oracle Database first tries to maximize the number of optimal work areas for all memory-intensive SQL operations. The rest of the work areas are executed in one-pass mode, unless the PGA memory limit set by the DBA (using the `PGA_AGGREGATE_TARGET` parameter) is so low that multi-pass execution is required to reduce memory consumption to honor the PGA target limit.

When configuring a new database instance, it can be difficult to determine the appropriate setting for `PGA_AGGREGATE_TARGET`.

To configure automatic PGA memory management:

1. Make an initial estimate for the value of the `PGA_AGGREGATE_TARGET` parameter, as described in "Setting the Initial Value for PGA_AGGREGATE_TARGET".
2. Run a representative workload on the database instance and monitor its performance, as described in "Monitoring Automatic PGA Memory Management".
3. Tune the value of the `PGA_AGGREGATE_TARGET` parameter using Oracle PGA advice statistics, as described in "Tuning PGA_AGGREGATE_TARGET".

> **See Also:**
>
> *Oracle Database Reference* for information about the `PGA_AGGREGATE_TARGET` initialization parameter

Setting the Initial Value for PGA_AGGREGATE_TARGET

Set the initial value of the `PGA_AGGREGATE_TARGET` initialization parameter based on the amount of available memory for the Oracle database instance. This value can then be tuned and dynamically modified at the instance level. By default, Oracle Database uses 20% of the SGA size for this value. However, this setting may be too low for a large DSS system.

To set the initial value for PGA_AGGREGATE_TARGET:

1. Determine how much of the total physical memory to reserve for the operating system and other non-Oracle applications running on the same system.

   For example, you might decide to reserve 20% of the total physical memory for the operating system and other non-Oracle applications, dedicating 80% of the memory on the system to the Oracle database instance.

2. Divide the remaining available memory between the SGA and the PGA:

   - For OLTP systems, the PGA memory typically makes up a small fraction of the available memory, leaving most of the remaining memory for the SGA.

     Oracle recommends initially dedicating 20% of the available memory to the PGA, and 80% to the SGA. Therefore, the initial value of the `PGA_AGGREGATE_TARGET` parameter for an OLTP system can be calculated as:

     `PGA_AGGREGATE_TARGET` = (*`total_mem`* * 0.8) * 0.2 where *`total_mem`* is the total amount of physical memory available on the system.

   - For DSS systems running large, memory-intensive queries, PGA memory can typically use up to 70% of the available memory.

     Oracle recommends initially dedicating 50% of the available memory to the PGA, and 50% to the SGA. Therefore, the initial value of the `PGA_AGGREGATE_TARGET` parameter for a DSS system can be calculated as:

     `PGA_AGGREGATE_TARGET` = (*`total_mem`* * 0.8) * 0.5 where *`total_mem`* is the total amount of physical memory available on the system.

For example, if an Oracle database instance is configured to run on a system with 4 GB of physical memory, and if 80% (or 3.2 GB) of the memory is dedicated to the Oracle database instance, then initially set `PGA_AGGREGATE_TARGET` to 640 MB for an OLTP system, or 1,600 MB for a DSS system.

Monitoring Automatic PGA Memory Management

Before starting the tuning process, run a representative workload on the database instance and monitor its performance. PGA statistics collected by Oracle Database enable you to determine if the maximum PGA size is under-configured or over-configured. Monitoring these statistics enables you to assess the performance of

automatic PGA memory management and tune the value of the PGA_AGGREGATE_TARGET parameter accordingly.

This section describes how to use performance views to monitor automatic PGA memory management and contains the following topics:

- Using the V$PGASTAT View
- Using the V$PROCESS View
- Using the V$PROCESS_MEMORY View
- Using the V$SQL_WORKAREA_HISTOGRAM View
- Using the V$WORKAREA_ACTIVE View
- Using the V$SQL_WORKAREA View

Using the V$PGASTAT View

The V$PGASTAT view provides instance-level statistics about PGA memory usage and the automatic PGA memory manager.

The following example shows a query of this view.

```
SELECT *
  FROM V$PGASTAT;
```

The output of this query might look like the following:

```
NAME                                                       VALUE UNIT
---------------------------------------------------------- ---------- ------------
aggregate PGA target parameter                           41156608 bytes
aggregate PGA auto target                                21823488 bytes
global memory bound                                       2057216 bytes
total PGA inuse                                          16899072 bytes
total PGA allocated                                      35014656 bytes
maximum PGA allocated                                   136795136 bytes
total freeable PGA memory                                 524288 bytes
PGA memory freed back to OS                             1713242112 bytes
total PGA used for auto workareas                             0 bytes
maximum PGA used for auto workareas                     2383872 bytes
total PGA used for manual workareas                           0 bytes
maximum PGA used for manual workareas                    8470528 bytes
over allocation count                                        291
bytes processed                                         2124600320 bytes
extra bytes read/written                                  39949312 bytes
cache hit percentage                                         98.15 percent
```

Table 16-1 describes the main statistics shown in the V$PGASTAT view.

Table 16-1 Statistics in the V$PGASTAT View

Statistic	Description
aggregate PGA target parameter	This statistic shows the current value of the PGA_AGGREGATE_TARGET parameter. The default value is 20% of the SGA size. Setting this parameter to 0 disables automatic PGA memory management.

Table 16-1 (Cont.) Statistics in the V$PGASTAT View

Statistic	Description
aggregate PGA auto target	This statistic shows the amount of PGA memory Oracle Database can use for work areas running in automatic mode. This amount is dynamically derived from the value of the PGA_AGGREGATE_TARGET parameter and the current work area workload. Hence, it is continuously adjusted by Oracle Database. If this value is small compared to the PGA_AGGREGATE_TARGET value, then most of PGA memory is used by other system components (such as PL/SQL or Java) and little is left for work areas. Ensure that enough PGA memory remains for work areas running in automatic mode.
global memory bound	This statistic shows the maximum size of a work area executed in automatic mode. This value is continuously adjusted by Oracle Database to reflect the current state of the work area workload. The global memory bound generally decreases when the number of active work areas increases in the system. As a rule of thumb, the value of the global bound should not decrease to less than 1 MB. If it does, increase the value of the PGA_AGGREGATE_TARGET parameter.
total PGA allocated	This statistic shows the current amount of PGA memory allocated by the database instance. Oracle Database tries to keep this number less than the PGA_AGGREGATE_TARGET value. However, if the work area workload is increasing rapidly or the PGA_AGGREGATE_TARGET parameter is set to a value that is too low, it is possible for the PGA allocated to exceed this value by a small percentage and for a short time.
total freeable PGA memory	This statistic indicates how much allocated PGA memory can be freed.
total PGA used for auto workareas	This statistic indicates how much PGA memory is currently consumed by work areas running in automatic mode. Use this number to determine how much memory is consumed by other consumers of the PGA memory (such as PL/SQL or Java): `PGA other = total PGA allocated - total PGA used for auto workareas`
over allocation count	This statistic is cumulative from instance startup. Over-allocating PGA memory can happen if the PGA_AGGREGATE_TARGET value is too small to accommodate the PGA other component and the minimum memory required to execute the work area workload. In this case, Oracle Database cannot honor the PGA_AGGREGATE_TARGET value, and extra PGA memory must be allocated. If over-allocation occurs, increase the value of the PGA_AGGREGATE_TARGET parameter using the information provided by the V$PGA_TARGET_ADVICE view, as described in "Using the V$PGA_TARGET_ADVICE View".
total bytes processed	This statistic indicates the number of bytes processed by memory-intensive SQL operators since instance startup. For example, the number of bytes processed is the input size for a sort operation. This number is used to compute the cache hit percentage metric.
extra bytes read/ written	When a work area cannot run optimally, one or more extra passes is performed over the input data. This statistic represents the number of bytes processed during these extra passes since instance startup. This number is also used to compute the cache hit percentage metric. Ideally, it should be small compared to total bytes processed.

Table 16-1 (Cont.) Statistics in the V$PGASTAT View

Statistic	Description
cache hit percentage	This metric is computed by Oracle Database to reflect the performance of the PGA memory component. It is cumulative from instance startup. A value of 100% means that all work areas executed by the system since instance startup are using an optimal amount of PGA memory. This is ideal but rarely happens except for pure OLTP systems. Typically, some work areas run one-pass or even multi-pass, depending on the overall size of the PGA memory. When a work area cannot run optimally, one or more extra passes are performed over the input data. This reduces the cache hit percentage in proportion to the size of the input data and the number of extra passes performed. For an example of how this metric is calculated, see Example 16-1.

Example 16-1 shows how extra passes affect the cache hit percentage metric.

Example 16-1 Calculating Cache Hit Percentage

Four sort operations have been executed, three were small (1 MB of input data) and one was bigger (100 MB of input data). The total number of bytes processed (BP) by the four operations is 103 MB. If one of the small sorts runs one-pass, an extra pass over 1 MB of input data is performed. This 1 MB value is the number of extra bytes read/written, or EBP.

The cache hit percentage is calculated using the following formula:

```
BP x 100 / (BP + EBP)
```

In this example, the cache hit percentage is 99.03%. This value reflects that only one of the small sort operations performed an extra pass, while all other sort operations were able to run in optimal size. Therefore, the cache hit percentage is almost 100%, because the extra pass over 1 MB represents a tiny overhead. However, if the bigger sort operation runs in one-pass size, then the EBP is 100 MB instead of 1 MB, and the cache hit percentage falls to 50.73%, because the extra pass has a much bigger impact.

Using the V$PROCESS View

The V$PROCESS view contains one row for each Oracle process connected to the database instance. Use the following columns in this view to monitor the PGA memory usage of these processes:

- PGA_USED_MEM
- PGA_ALLOC_MEM
- PGA_FREEABLE_MEM
- PGA_MAX_MEM

Example 16-2 shows a query of this view.

Example 16-2 Querying the V$PROCESS View

```
SELECT program, pga_used_mem, pga_alloc_mem, pga_freeable_mem, pga_max_mem
  FROM V$PROCESS;
```

The output of this query might look like the following:

```
PROGRAM                  PGA_USED_MEM PGA_ALLOC_MEM PGA_FREEABLE_MEM PGA_MAX_MEM
------------------------ ------------ ------------- ---------------- -----------
PSEUDO                              0             0                0           0
oracle@examp1690 (PMON)        314540        685860                0      685860
oracle@examp1690 (MMAN)        313992        685860                0      685860
oracle@examp1690 (DBW0)        696720       1063112                0     1063112
oracle@examp1690 (LGWR)      10835108      22967940                0    22967940
oracle@examp1690 (CKPT)        352716        710376                0      710376
oracle@examp1690 (SMON)        541508        948004                0     1603364
oracle@examp1690 (RECO)        323688        685860                0      816932
oracle@examp1690 (q001)        233508        585128                0      585128
oracle@examp1690 (QMNC)        314332        685860                0      685860
oracle@examp1690 (MMON)        885756       1996548           393216     1996548
oracle@examp1690 (MMNL)        315068        685860                0      685860
oracle@examp1690 (q000)        330872        716200            65536      716200
oracle@examp1690 (CJQ0)        533476       1013540                0     1144612
```

Using the V$PROCESS_MEMORY View

The `V$PROCESS_MEMORY` view displays dynamic PGA memory usage by named component categories for each Oracle process. This view contains up to six rows for each Oracle process, one row for:

- Each named component category:
  - Java
  - PL/SQL
  - OLAP
  - SQL
- Freeable

  Memory that has been allocated to the process by the operating system, but not to a specific category
- Other

  Memory that has been allocated to a category, but not to a named category

Use the following columns in this view to dynamically monitor the PGA memory usage of Oracle processes for each of the six categories:

- `CATEGORY`
- `ALLOCATED`
- `USED`
- `MAX_ALLOCATED`

> **Note:**
>
> The `V$PROCESS_MEMORY_DETAIL` view displays dynamic PGA memory usage for the Oracle processes that exceed 500 MB of PGA usage. The `V$PROCESS_MEMORY_DETAIL` view is available starting with Oracle Database 12*c* Release 2.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `V$PROCESS_MEMORY` and `V$PROCESS_MEMORY_DETAIL` views

Using the V$SQL_WORKAREA_HISTOGRAM View

The `V$SQL_WORKAREA_HISTOGRAM` view shows the number of work areas executed with optimal, one-pass, and multi-pass memory size since instance startup. Statistics in this view are divided into buckets. The buckets are defined by the optimal memory requirements of the work areas. Each bucket is identified by a range of optimal memory requirements, specified by the values in the `LOW_OPTIMAL_SIZE` and `HIGH_OPTIMAL_SIZE` columns.

For example, a sort operation may require 3 MB of memory to run in optimal size (cached). Statistics about the work area used by this sort operation are placed in the bucket defined by:

- `LOW_OPTIMAL_SIZE` = 2097152 (2 MB)
- `HIGH_OPTIMAL_SIZE` = 4194303 (4 MB minus 1 byte)

Statistics are segmented by work area size, because the performance impact of running a work area in optimal, one-pass or multi-pass size depends mainly on the size of the work area. In this example, statistics about the work area are placed in this bucket because 3 MB lies within that range of optimal sizes.

Example 16-3 and Example 16-4 show two methods for querying this view.

Example 16-3 Querying the V$SQL_WORKAREA_HISTOGRAM View: Non-Empty Buckets

The following query shows statistics for all non-empty buckets:

```
SELECT low_optimal_size/1024 low_kb,
       (high_optimal_size+1)/1024 high_kb,
       optimal_executions, onepass_executions, multipasses_executions
  FROM V$SQL_WORKAREA_HISTOGRAM
 WHERE total_executions != 0;
```

The result of the query might look like the following:

```
LOW_KB HIGH_KB OPTIMAL_EXECUTIONS ONEPASS_EXECUTIONS MULTIPASSES_EXECUTIONS
------ ------- ------------------ ------------------ ----------------------
     8      16             156255                  0                      0
    16      32                150                  0                      0
    32      64                 89                  0                      0
    64     128                 13                  0                      0
   128     256                 60                  0                      0
   256     512                  8                  0                      0
   512    1024                657                  0                      0
  1024    2048                551                 16                      0
  2048    4096                538                 26                      0
  4096    8192                243                 28                      0
  8192   16384                137                 35                      0
 16384   32768                 45                107                      0
 32768   65536                  0                153                      0
 65536  131072                  0                 73                      0
131072  262144                  0                 44                      0
262144  524288                  0                 22                      0
```

In this example, the output shows that—in the 1 MB to 2 MB bucket—551 work areas ran in optimal size, while 16 ran in one-pass size and none ran in multi-pass size. It also shows that all work areas under 1 MB were able to run in optimal size.

Example 16-4 Querying the V$SQL_WORKAREA_HISTOGRAM View: Percent Optimal

The following query shows the percentage of times work areas are executed in optimal, one-pass, or multi-pass size since startup. This query only considers work areas of a certain size, with an optimal memory requirement of at least 64 KB:

```
SELECT optimal_count, ROUND(optimal_count*100/total, 2) optimal_perc,
       onepass_count, ROUND(onepass_count*100/total, 2) onepass_perc,
       multipass_count, ROUND(multipass_count*100/total, 2) multipass_perc
FROM
 (SELECT DECODE(SUM(total_executions), 0, 1, SUM(total_executions)) total,
         SUM(optimal_executions) optimal_count,
         SUM(onepass_executions) onepass_count,
         SUM(multipass_executions) multipass_count
    FROM V$SQL_WORKAREA_HISTOGRAM
   WHERE low_optimal_size >= 64*1024);
```

The output of this query might look like the following:

```
OPTIMAL_COUNT OPTIMAL_PERC ONEPASS_COUNT ONEPASS_PERC MULTIPASS_COUNT
MULTIPASS_PERC
------------- ------------ ------------- ------------ ---------------
--------------
         2239        81.63           504        18.37
0              0
```

In this example, the output shows that 81.63% of the work areas were able to run in optimal size. The remaining work areas (18.37%) ran in one-pass size and none of them ran in multi-pass size.

Using the V$WORKAREA_ACTIVE View

The `V$WORKAREA_ACTIVE` view displays the work areas that are active (or executing) in the database instance. Small, active sort operations (under 64 KB) are excluded from this view. Use this view to precisely monitor the size of all active work areas and to determine whether these active work areas spill to a temporary segment.

Example 16-5 shows a query of this view.

Example 16-5 Querying the V$WORKAREA_ACTIVE View

```
SELECT TO_NUMBER(DECODE(sid, 65535, null, sid)) sid,
       operation_type operation,
       TRUNC(expected_size/1024) esize,
       TRUNC(actual_mem_used/1024) mem,
       TRUNC(max_mem_used/1024) "max mem",
       number_passes pass,
       TRUNC(TEMPSEG_SIZE/1024) tsize
  FROM V$SQL_WORKAREA_ACTIVE
 ORDER BY 1,2;
```

The output of this query might look like the following:

```
SID         OPERATION     ESIZE       MEM   MAX MEM  PASS   TSIZE
--- ----------------- --------- --------- --------- ----- -------
```

```
  8   GROUP BY (SORT)       315       280       904     0
  8         HASH-JOIN      2995      2377      2430     1   20000
  9   GROUP BY (SORT)      34300     22688     22688     0
 11         HASH-JOIN      18044     54482     54482     0
 12         HASH-JOIN      18044     11406     21406     1  120000
```

In this example, the output shows that:

- Session 12 (`SID` column) is running a hash-join operation (`OPERATION` column) in a work area running in one-pass size (`PASS` column)
- The maximum amount of memory that the PGA memory manager expects this hash-join operation to use is 18044 KB (`ESIZE` column)
- The work area is currently using 11406 KB of memory (`MEM` column)
- The work area used up to 21406 KB of PGA memory (`MAX MEM` column) in the past
- The work area spilled to a temporary segment of 120000 KB (`TSIZE` column)

When the work area is deallocated—or when the execution of its associated SQL operator is complete—it is automatically removed from this view.

Using the V$SQL_WORKAREA View

Oracle Database maintains cumulative work area statistics for each loaded cursor whose execution plan uses one or more work areas. Each time a work area is deallocated, the `V$SQL_WORKAREA` view is updated with execution statistics for that work area.

You can join the `V$SQL_WORKAREA` view with the `V$SQL` view to relate a work area to a cursor, and with the `V$SQL_PLAN` view to precisely determine which operator in the plan uses a work area.

Example 16-6 shows three queries of this view.

Example 16-6 Querying the V$SQL_WORKAREA View

The following query finds the top 10 work areas that require the most cache memory:

```
SELECT *
FROM   (SELECT workarea_address, operation_type, policy, estimated_optimal_size
        FROM V$SQL_WORKAREA
        ORDER BY estimated_optimal_size DESC)
 WHERE ROWNUM <= 10;
```

The following query finds the cursors with one or more work areas that have been executed in one or multiple passes:

```
col sql_text format A80 wrap
SELECT sql_text, sum(ONEPASS_EXECUTIONS) onepass_cnt,
       sum(MULTIPASSES_EXECUTIONS) mpass_cnt
FROM V$SQL s, V$SQL_WORKAREA wa
WHERE s.address = wa.address
GROUP BY sql_text
HAVING sum(ONEPASS_EXECUTIONS+MULTIPASSES_EXECUTIONS)>0;
```

Using the hash value and address of a particular cursor, the following query displays the cursor execution plan, including information about the associated work areas:

```
col "O/1/M" format a10
col name format a20
SELECT operation, options, object_name name, trunc(bytes/1024/1024) "input(MB)",
```

```
       TRUNC(last_memory_used/1024) last_mem,
       TRUNC(estimated_optimal_size/1024) optimal_mem,
       TRUNC(estimated_onepass_size/1024) onepass_mem,
       DECODE(optimal_executions, null, null,
              optimal_executions||'/'||onepass_executions||'/'||
              multipasses_executions) "O/1/M"
  FROM V$SQL_PLAN p, V$SQL_WORKAREA w
 WHERE p.address=w.address(+)
   AND p.hash_value=w.hash_value(+)
   AND p.id=w.operation_id(+)
   AND p.address='88BB460C'
   AND p.hash_value=3738161960;
```

The output of this query might look like the following:

```
OPERATION    OPTIONS  NAME     input(MB) LAST_MEM OPTIMAL_ME ONEPASS_ME O/1/M
------------ -------- -------- --------- -------- ---------- ---------- ------
SELECT STATE
HASH         GROUP BY               4582        8         16         16 16/0/0
HASH JOIN    SEMI                   4582     5976       5194       2187 16/0/0
TABLE ACCESS FULL     ORDERS           51
TABLE ACCESS FUL      LINEITEM       1000
```

You can get the address and hash value from the `V$SQL` view by specifying a pattern in the query, as shown in the following query:

```
SELECT address, hash_value
  FROM V$SQL
 WHERE sql_text LIKE '%my_pattern%';
```

Tuning PGA_AGGREGATE_TARGET

To help you tune the value of the `PGA_AGGREGATE_TARGET` initialization parameter, Oracle Database provides two PGA performance advisory views: `V$PGA_TARGET_ADVICE` and `V$PGA_TARGET_ADVICE_HISTOGRAM`. By using these views, you do not need to use an empirical approach to tune the value of the `PGA_AGGREGATE_TARGET` parameter. Instead, you can use these views to predict how changing the value of the `PGA_AGGREGATE_TARGET` parameter will affect key PGA statistics.

This section describes how to tune the value of the `PGA_AGGREGATE_TARGET` initialization parameter and contains the following topics:

- Enabling Automatic Generation of PGA Performance Advisory Views
- Using the V$PGA_TARGET_ADVICE View
- Using the V$PGA_TARGET_ADVICE_HISTOGRAM View
- Using the V$SYSSTAT and V$SESSTAT Views
- Tutorial: How to Tune PGA_AGGREGATE_TARGET

Enabling Automatic Generation of PGA Performance Advisory Views

Oracle Database generates the `V$PGA_TARGET_ADVICE` and `V$PGA_TARGET_ADVICE_HISTOGRAM` views by recording the workload history, and then simulating this history for different values of the `PGA_AGGREGATE_TARGET` parameter. The values of the `PGA_AGGREGATE_TARGET` parameter are derived from fractions and

multiples of its current value to assess possible higher and lower values. These values are used for the prediction and range from 10 MB to a maximum of 256 GB. The simulation process happens in the background and continuously updates the workload history to produce the simulation result. You can view the result at any time by querying these views.

To enable automatic generation of PGA performance advice views:

1. Set the `PGA_AGGREGATE_TARGET` parameter to enable automatic PGA memory management.

   Setting this parameter to 0 disables automatic PGA memory management and is not recommended. For information about setting this parameter, see "Setting the Initial Value for PGA_AGGREGATE_TARGET".

2. Set the `STATISTICS_LEVEL` parameter to `TYPICAL` (the default) or `ALL`.

   Setting this parameter to `BASIC` disables generation of the PGA performance advice views and is not recommended.

> **Note:**
>
> The contents of the PGA advice performance views are reset at instance startup or when the value of the `PGA_AGGREGATE_TARGET` parameter is changed.

Using the V$PGA_TARGET_ADVICE View

The `V$PGA_TARGET_ADVICE` view predicts how changing the value of the `PGA_AGGREGATE_TARGET` initialization parameter will affect the following statistics in the `V$PGASTAT` view:

- cache hit percentage
- over allocation count

The following example shows a query of this view.

```
SELECT ROUND(pga_target_for_estimate/1024/1024) target_mb,
       estd_pga_cache_hit_percentage cache_hit_perc,
       estd_overalloc_count
  FROM V$PGA_TARGET_ADVICE;
```

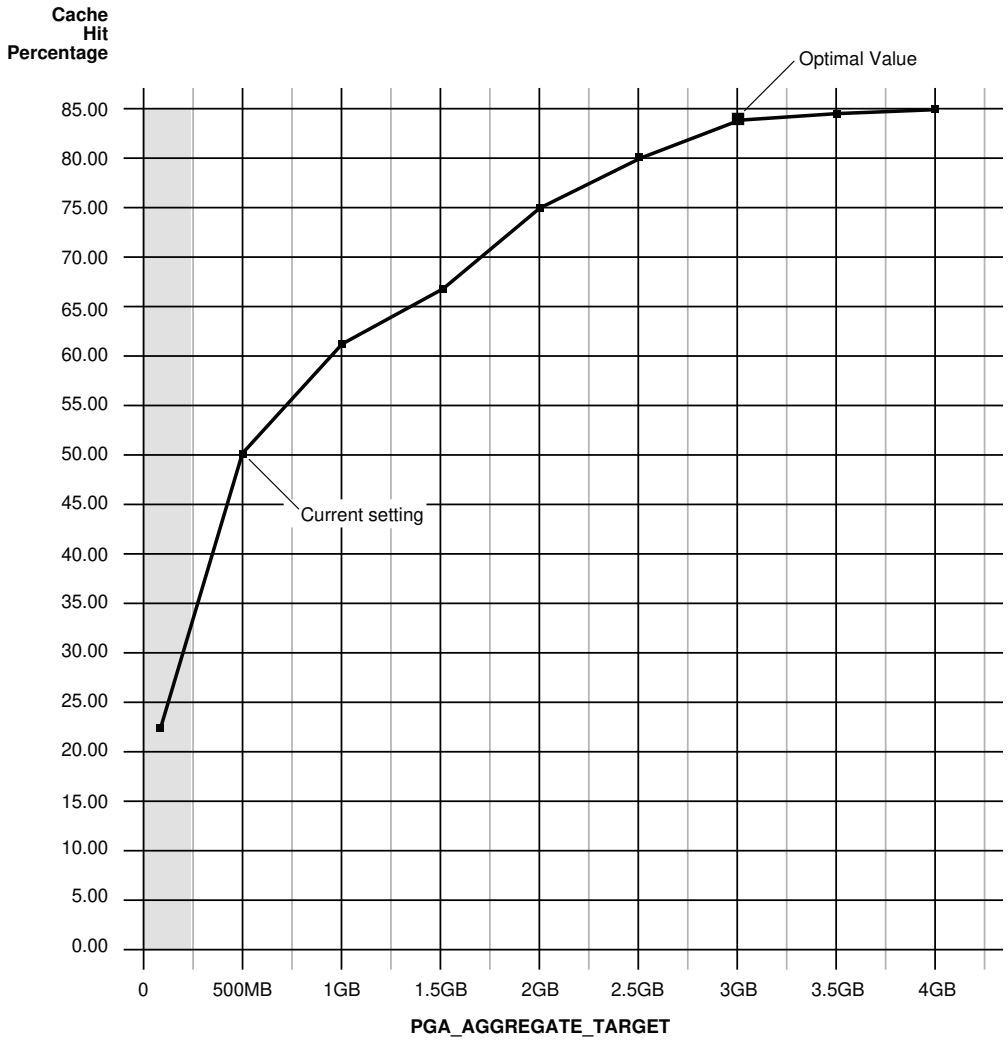
The output of this query might look like the following:

```
 TARGET_MB  CACHE_HIT_PERC  ESTD_OVERALLOC_COUNT
----------  --------------  --------------------
        63              23                   367
       125              24                    30
       250              30                     3
       375              39                     0
       500              58                     0
       600              59                     0
       700              59                     0
       800              60                     0
       900              60                     0
      1000              61                     0
      1500              67                     0
      2000              76                     0
```

```
3000                83                  0
4000                85                  0
```

The following figure illustrates how the result of this query can be plotted.

Figure 16-1 Graphical Representation of V$PGA_TARGET_ADVICE



The curve shows how PGA `cache hit percentage` improves as the value of the `PGA_AGGREGATE_TARGET` parameter increases. The shaded zone in the graph represents the `over allocation` zone, where the value of the `ESTD_OVERALLOCATION_COUNT` column is non-zero. This area indicates that the value of the `PGA_AGGREGATE_TARGET` parameter is too small to meet the minimum PGA memory requirements. If the value of the `PGA_AGGREGATE_TARGET` parameter is set within the `over allocation` zone, then the memory manager will over-allocate memory and the actual PGA memory consumed will exceed the limit that was set. It is therefore meaningless to set a value of the `PGA_AGGREGATE_TARGET` parameter in that zone. In this particular example, the `PGA_AGGREGATE_TARGET` parameter should be set to at least 375 MB.

Beyond the over allocation zone, the value of the PGA cache hit percentage increases rapidly. This is due to an increase in the number of optimal or one-pass work areas and a decrease in the number of multi-pass executions. At some point, around 500 MB in this example, an inflection in the curve corresponds to the point where most (probably all) work areas can run in optimal or at least one-pass size. Beyond this point, the cache hit percentage keeps increasing, though at a lower pace, up to the point where it starts to taper off and only slight improvement is achieved with increase in the value of the PGA_AGGREGATE_TARGET parameter. In the figure, this happens when PGA_AGGREGATE_TARGET reaches 3 GB. At this point, the cache hit percentage is 83% and only marginal improvement (by 2%) is achieved with one extra gigabyte of PGA memory. In this example, 3 GB is the optimal value for the PGA_AGGREGATE_TARGET parameter.

> **Note:**
>
> Although the theoretical maximum for the PGA cache hit percentage is 100%, a practical limit exists on the maximum size of a work area that may prevent this theoretical maximum from being reached, even when the value of the PGA_AGGREGATE_TARGET parameter is further increased. This should happen only in large DSS systems where the optimal memory requirement is large and may cause the value of the cache hit percentage to taper off at a lower percentage, such as 90%.

Ideally, the value of the PGA_AGGREGATE_TARGET parameter should be set to the optimal value, or at least to the maximum value possible in the region beyond the over allocation zone. As a rule of thumb, the PGA cache hit percentage should be higher than 60%, because at 60% the system is almost processing double the number of bytes it actually needs to process in an ideal situation. In this example, the value of the PGA_AGGREGATE_TARGET parameter should be set to at least 500 MB, and as close to 3 GB as possible. However, the correct setting for the PGA_AGGREGATE_TARGET parameter depends on how much memory can be dedicated to the PGA component. Generally, adding PGA memory requires reducing memory for some SGA components—like the shared pool or buffer cache—because the overall memory dedicated to the database instance is often bound by the amount of physical memory available on the system. Therefore, any decisions to increase PGA memory must be taken in the larger context of the available memory in the system and the performance of the various SGA components (which you can monitor with shared pool advisory and buffer cache advisory statistics). If you cannot reduce memory from the SGA components, consider adding more physical memory to the system.

Using the V$PGA_TARGET_ADVICE_HISTOGRAM View

The V$PGA_TARGET_ADVICE_HISTOGRAM view predicts how changing the value of the PGA_AGGREGATE_TARGET initialization parameter will affect the statistics in the V$SQL_WORKAREA_HISTOGRAM view. Use this view to display detailed information about the predicted number of optimal, one-pass, and multi-pass work area executions for the PGA_AGGREGATE_TARGET values used for the prediction.

The V$PGA_TARGET_ADVICE_HISTOGRAM view is identical to the V$SQL_WORKAREA_HISTOGRAM view, with two additional columns to represent the PGA_AGGREGATE_TARGET values used for the prediction. Therefore, any query executed against the V$SQL_WORKAREA_HISTOGRAM view can be used on this view, with an additional predicate to select the desired value of the PGA_AGGREGATE_TARGET parameter.

Example 16-7 shows a query of this view that displays the predicted content of the V$SQL_WORKAREA_HISTOGRAM view for a value of the PGA_AGGREGATE_TARGET parameter set to twice its current value.

Example 16-7 Querying the V$PGA_TARGET_ADVICE_HISTOGRAM View

```
SELECT low_optimal_size/1024 low_kb, (high_optimal_size+1)/1024 high_kb,
       estd_optimal_executions estd_opt_cnt,
       estd_onepass_executions estd_onepass_cnt,
       estd_multipasses_executions estd_mpass_cnt
  FROM V$PGA_TARGET_ADVICE_HISTOGRAM
 WHERE pga_target_factor = 2
   AND estd_total_executions != 0
 ORDER BY 1;
```

The output of this query might look like the following:

```
LOW_KB   HIGH_KB   ESTD_OPTIMAL_CNT   ESTD_ONEPASS_CNT   ESTD_MPASS_CNT
------   -------   ----------------   ----------------   --------------
     8        16             156107                  0                0
    16        32                148                  0                0
    32        64                 89                  0                0
    64       128                 13                  0                0
   128       256                 58                  0                0
   256       512                 10                  0                0
   512      1024                653                  0                0
  1024      2048                530                  0                0
  2048      4096                509                  0                0
  4096      8192                227                  0                0
  8192     16384                176                  0                0
 16384     32768                133                 16                0
 32768     65536                 66                103                0
 65536    131072                 15                 47                0
131072    262144                  0                 48                0
262144    524288                  0                 23                0
```

In this example, the output shows that increasing the value of the PGA_AGGREGATE_TARGET parameter by a factor of 2 will enable all work areas under 16 MB to execute in optimal size.

Using the V$SYSSTAT and V$SESSTAT Views

Statistics in the V$SYSSTAT and V$SESSTAT views show the total number of work areas executed with optimal, one-pass, and multi-pass memory size. These statistics are cumulative since the instance or the session was started.

Example 16-8 shows a query of the V$SYSSTAT view that displays the total number and the percentage of times work areas were executed in these three sizes since the instance was started:

Example 16-8 Querying the V$SYSSTAT View

```
SELECT name profile, cnt, DECODE(total, 0, 0, ROUND(cnt*100/total)) percentage
  FROM (SELECT name, value cnt, (SUM(value) over ()) total
  FROM V$SYSSTAT
 WHERE name
  LIKE 'workarea exec%');
```

The output of this query might look like the following:

```
PROFILE                                    CNT PERCENTAGE
---------------------------------- ---------- ----------
workarea executions - optimal             5395         95
workarea executions - onepass              284          5
workarea executions - multipass              0          0
```

In this example, the output shows that 5,395 work area executions (or 95%) were executed in optimal size, and 284 work area executions (or 5%) were executed in one-pass size.

Tutorial: How to Tune PGA_AGGREGATE_TARGET

This tutorial provides a guideline for tuning the value of the `PGA_AGGREGATE_TARGET` parameter using the various views discussed in this chapter.

To tune PGA_AGGREGATE_TARGET:

1. Set the value of the `PGA_AGGREGATE_TARGET` parameter to avoid memory over-allocation.

   Use the `V$PGA_TARGET_ADVICE` view to ensure that the `PGA_AGGREGATE_TARGET` value is not set within the over-allocation zone, as described in "Using the V$PGA_TARGET_ADVICE View". In Example 16–8, the `PGA_AGGREGATE_TARGET` value should be set to at least 375 MB.

2. Maximize the PGA `cache hit percentage`, based on response time requirements and memory constraints.

   Use the `V$PGA_TARGET_ADVICE` view to determine the optimal value for the `PGA_AGGREGATE_TARGET` parameter and set its value to the optimal value, or to the maximum value possible, as described in "Using the V$PGA_TARGET_ADVICE View".

   Assume a limit *X* on the memory that can be allocated to PGA:

   - If limit *X* is higher than the optimal value, set the value of the `PGA_AGGREGATE_TARGET` parameter to the optimal value.

     In Example 16–8, if you have 10 GB to dedicate to PGA, set the value of the `PGA_AGGREGATE_TARGET` parameter to 3 GB and dedicate the remaining 7 GB to the SGA.

   - If limit *X* is less than the optimal value, set the value of the `PGA_AGGREGATE_TARGET` parameter to *X*.

     In Example 16–8, if you have only 2 GB to dedicate to PGA, set the value of the `PGA_AGGREGATE_TARGET` parameter to 2 GB and accept a `cache hit percentage` of 75%.

3. Verify that the new value of the `PGA_AGGREGATE_TARGET` parameter will result in the desired number of optimal and one-pass work area executions and avoid any multi-pass work area executions.

   Use the `V$PGA_TARGET_ADVICE_HISTOGRAM` view to predict the number of optimal, one-pass, and multi-pass work area executions, as described in "Using the V$PGA_TARGET_ADVICE_HISTOGRAM View".

4. If more PGA memory is required, then increase PGA memory by either reducing memory from SGA components or adding more physical memory to the system.

5. At any time, ensure the number of optimal, one-pass, and multi-pass work area executions matches predictions and tune the value of the `PGA_AGGREGATE_TARGET` parameter if necessary.

Use the V$SYSSTAT and V$SESSTAT views to verify the total number of work areas executed with optimal, one-pass, and multi-pass memory size since instance or session startup, respectively, as described in "Using the V$SYSSTAT and V$SESSTAT Views".

Sizing the Program Global Area by Specifying an Absolute Limit

In automatic PGA memory management mode, Oracle Database attempts to adhere to the PGA_AGGREGATE_TARGET value by dynamically controlling the amount of PGA memory allotted to work areas. However, PGA memory usage may exceed the PGA_AGGREGATE_TARGET setting at times due to the following reasons:

- The PGA_AGGREGATE_TARGET setting acts as a target, and not a limit.
- PGA_AGGREGATE_TARGET only controls allocations of tunable memory.

Excessive PGA usage can lead to high rates of swapping. When this occurs, the system may become unresponsive and unstable. In that case, consider using any of the following methods to specify an absolute limit on the PGA memory usage:

- Use PGA_AGGREGATE_LIMIT parameter to set an absolute limit on the overall PGA memory usage.

  See "Sizing the Program Global Area Using the PGA_AGGREGATE_LIMIT Parameter"
- Use the Resource Manager procedure DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE to set an absolute limit on the PGA memory usage for each session in a particular consumer group.

  See "Sizing the Program Global Area Using the Resource Manager"

Sizing the Program Global Area Using the PGA_AGGREGATE_LIMIT Parameter

The PGA_AGGREGATE_LIMIT initialization parameter enables you to specify an absolute limit on the PGA memory usage. If the PGA_AGGREGATE_LIMIT value is exceeded, Oracle Database aborts or terminates the sessions or processes that are consuming the most untunable PGA memory in the following order:

- Calls for sessions that are consuming the most untunable PGA memory are aborted.
- If PGA memory usage is still over the PGA_AGGREGATE_LIMIT, then the sessions and processes that are consuming the most untunable PGA memory are terminated.

In determining the sessions and processes to abort or terminate, Oracle Database treats parallel queries as a single unit.

By default, the PGA_AGGREGATE_LIMIT parameter is set to the greater of 2 GB, 200% of the PGA_AGGREGATE_TARGET value, or 3 MB times the value of the PROCESSES parameter. However, it will not exceed 120% of the physical memory size minus the total SGA size. The default value is printed into the alert log. A warning message is

printed in the alert log if the amount of physical memory on the system cannot be determined.

To set PGA_AGGREGATE_LIMIT:

- Set the `PGA_AGGREGATE_LIMIT` initialization parameter to a desired value in number of bytes.

  The value is expresses as a number followed by `K` (for kilobytes), `M` (for megabytes), or `G` (for gigabytes). Setting the value to 0 disables the hard limit on PGA memory.

> **See Also:**
>
> - *Oracle Database Reference* for information about the `PGA_AGGREGATE_LIMIT` initialization parameter
> - *Oracle Database Reference* for information about the `V$PGASTAT` view
> - *Oracle Database Administrator's Guide* for information about Oracle Database Resource Manager and consumer groups

Sizing the Program Global Area Using the Resource Manager

You can set an absolute limit on the amount of PGA memory that can be allocated to each session in a particular consumer group using the `SESSION_PGA_LIMIT` parameter of the `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE` procedure of the Oracle Database Resource Manager. If a session exceeds the PGA memory limit set for its consumer group, then that session is terminated with the `ORA-10260` error message.

> **See Also:**
>
> - *Oracle Database Administration Guide* topics:
>   - "Program Global Area (PGA)" for more information about limiting the PGA memory for each session in a consumer group.
>   - "Creating Resource Plan Directives" for more information about creating resource plan directives using the `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE` procedure.
> - *Oracle Database PL/SQL Packages and Types Reference* for the syntax of the `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE` procedure.

Part IV
Managing System Resources

This part contains the following chapters:

17

I/O Configuration and Design

The I/O subsystem is a vital component of an Oracle database. This chapter introduces fundamental I/O concepts, discusses the I/O requirements of different parts of the database, and provides sample configurations for I/O subsystem design.

This chapter includes the following topics:

About I/O

Every Oracle database reads or writes data on disk, thus generating **disk I/O**. The performance of many software applications is inherently limited by disk I/O. Applications that spend majority of their CPU time waiting for I/O activity to complete are said to be **I/O-bound**.

Oracle Database is designed so that if an application is well written, its performance should not be limited by I/O. Tuning I/O can enhance the performance of the application if the I/O system is operating at or near capacity and is not able to service the I/O requests within an acceptable time. However, tuning I/O cannot help performance if the application is not I/O-bound (for example, when CPU is the limiting factor).

Consider the following database requirements when designing an I/O system:

- Storage, such as minimum disk capacity
- Availability, such as continuous (24 x 7) or business hours only
- Performance, such as I/O throughput and application response times

Many I/O designs plan for storage and availability requirements with the assumption that performance will not be an issue. This is not always the case. Optimally, the number of disks and controllers to be configured should be determined by I/O throughput and redundancy requirements. The size of disks can then be determined by the storage requirements.

When developing an I/O design plan, consider using **Oracle Automatic Storage Management (Oracle ASM)**. Oracle ASM is an integrated, high-performance database file system and disk manager that is based on the principle that the database should manage storage instead of requiring an administrator to do it.

Oracle recommends that you use Oracle ASM for your database file storage, instead of raw devices or the operating system file system. Oracle ASM provides the following key benefits:

- Striping
- Mirroring
- Online storage reconfiguration and dynamic rebalancing
- Managed file creation and deletion

> **See Also:**
>
> *Oracle Automatic Storage Management Administrator's Guide* for additional information about Oracle ASM

I/O Configuration

This section describes the basic information to be gathered and decisions to be made when defining a system's I/O configuration. You want to keep the configuration as simple as possible, while maintaining the required availability, recoverability, and performance. The more complex a configuration becomes, the more difficult it is to administer, maintain, and tune.

This section contains the following topics:

- Lay Out the Files Using Operating System or Hardware Striping
- Manually Distributing I/O
- When to Separate Files
- Three Sample Configurations
- Oracle Managed Files
- Choosing Data Block Size

Lay Out the Files Using Operating System or Hardware Striping

If your operating system has LVM software or hardware-based striping, then it is possible to distribute I/O using these tools. Decisions to be made when using an LVM or hardware striping include **stripe depth** and **stripe width**.

- Stripe depth is the size of the stripe, sometimes called stripe unit.
- Stripe width is the product of the stripe depth and the number of drives in the striped set.

Choose these values wisely so that the system is capable of sustaining the required throughput. For an Oracle database, reasonable stripe depths range from 256 KB to 1 MB. Different types of applications benefit from different stripe depths. The optimal stripe depth and stripe width depend on the following:

- Requested I/O Size
- Concurrency of I/O Requests
- Alignment of Physical Stripe Boundaries with Block Size Boundaries
- Manageability of the Proposed System

Requested I/O Size

Table 17-1 lists the Oracle Database and operating system parameters that you can use to set I/O size:

Table 17-1 Oracle Database and Operating System Operational Parameters

Parameter	Description
DB_BLOCK_SIZE	The size of single-block I/O requests. This parameter is also used in combination with multiblock parameters to determine multiblock I/O request size.
OS block size	Determines I/O size for redo log and archive log operations.
Maximum OS I/O size	Places an upper bound on the size of a single I/O request.
DB_FILE_MULTIBLOCK_READ_COUNT	The maximum I/O size for full table scans is computed by multiplying this parameter with DB_BLOCK_SIZE. (the upper value is subject to operating system limits). If this value is not set explicitly (or is set to 0), the default value corresponds to the maximum I/O size that can be efficiently performed and is platform-dependent.
SORT_AREA_SIZE	Determines I/O sizes and concurrency for sort operations.
HASH_AREA_SIZE	Determines the I/O size for hash operations.

In addition to I/O size, the degree of concurrency also helps in determining the ideal stripe depth. Consider the following when choosing stripe width and stripe depth:

- On low-concurrency (sequential) systems, ensure that no single I/O visits the same disk twice. For example, assume that the stripe width is four disks, and the stripe depth is 32K. If a single 1MB I/O request (for example, for a full table scan) is issued by an Oracle server process, then each disk in the stripe must perform eight I/Os to return the requested data. To avoid this situation, the size of the average I/O should be smaller than the stripe width multiplied by the stripe depth. If this is not the case, then a single I/O request made by Oracle Database to the operating system results in multiple physical I/O requests to the same disk.
- On high-concurrency (random) systems, ensure that no single I/O request is broken up into multiple physical I/O calls. Failing to do this multiplies the number of physical I/O requests performed in your system, which in turn can severely degrade the I/O response times.

Concurrency of I/O Requests

In a system with a high degree of concurrent small I/O requests, such as in a traditional OLTP environment, it is beneficial to keep the stripe depth large. Using stripe depths larger than the I/O size is called **coarse grain striping**. In high-concurrency systems, the stripe depth can be as follows, where $n > 1$:

```
n * DB_BLOCK_SIZE
```

Coarse grain striping allows a disk in the array to service several I/O requests. In this way, a large number of concurrent I/O requests can be serviced by a set of striped disks with minimal I/O setup costs. Coarse grain striping strives to maximize overall I/O throughput. Multiblock reads, as in full table scans, will benefit when stripe depths are large and can be serviced from one drive. Parallel query in a data warehouse environment is also a candidate for coarse grain striping because many individual processes each issue separate I/Os. If coarse grain striping is used in systems that do not have high concurrent requests, then hot spots could result.

In a system with a few large I/O requests, such as in a traditional DSS environment or a low-concurrency OLTP system, then it is beneficial to keep the stripe depth small. This is called

fine grain striping. In such systems, the stripe depth is as follows, where *n* is smaller than the multiblock read parameters, such as `DB_FILE_MULTIBLOCK_READ_COUNT`:

```
n * DB_BLOCK_SIZE
```

Fine grain striping allows a single I/O request to be serviced by multiple disks. Fine grain striping strives to maximize performance for individual I/O requests or response time.

Alignment of Physical Stripe Boundaries with Block Size Boundaries

On some Oracle Database ports, a database block boundary may not align with the stripe. If your stripe depth is the same size as the database block, then a single I/O issued by Oracle Database may result in two physical I/O operations.

This is not optimal in an OLTP environment. To ensure a higher probability of one logical I/O resulting in no more than one physical I/O, the minimum stripe depth should be at least twice the Oracle block size. Table 17-2 shows recommended minimum stripe depth for random access and for sequential reads.

Table 17-2 Minimum Stripe Depth

Disk Access	Minimum Stripe Depth
Random reads and writes	The minimum stripe depth is twice the Oracle block size.
Sequential reads	The minimum stripe depth is twice the value of `DB_FILE_MULTIBLOCK_READ_COUNT`, multiplied by the Oracle block size.

> **See Also:**
>
> The specific documentation for your platform

Manageability of the Proposed System

With an LVM, the simplest configuration to manage is one with a single striped volume over all available disks. In this case, the stripe width encompasses all available disks. All database files reside within that volume, effectively distributing the load evenly. This single-volume layout provides adequate performance in most situations.

A single-volume configuration is viable only when used in conjunction with RAID technology that allows easy recoverability, such as RAID 1. Otherwise, losing a single disk means losing all files concurrently and, hence, performing a full database restore and recovery.

In addition to performance, there is a manageability concern: the design of the system must allow disks to be added simply, to allow for database growth. The challenge is to do so while keeping the load balanced evenly.

For example, an initial configuration can involve the creation of a single striped volume over 64 disks, each disk being 16 GB. This is total disk space of 1 terabyte (TB) for the primary data. Sometime after the system is operational, an additional 80 GB (that is, five disks) must be added to account for future database growth.

The options for making this space available to the database include creating a second volume that includes the five new disks. However, an I/O bottleneck might develop, if these new disks are unable to sustain the I/O throughput required for the files placed on them.

Another option is to increase the size of the original volume. LVMs are becoming sophisticated enough to allow dynamic reconfiguration of the stripe width, which allows disks to be added while the system is online. This begins to make the placement of all files on a single striped volume feasible in a production environment.

If your LVM cannot support dynamically adding disks to the stripe, then it is likely that you need to choose a smaller, more manageable stripe width. Then, when new disks are added, the system can grow by a stripe width.

In the preceding example, eight disks might be a more manageable stripe width. This is only feasible if eight disks are capable of sustaining the required number of I/Os each second. Thus, when extra disk space is required, another eight-disk stripe can be added, keeping the I/O balanced across the volumes.

> **Note:**
>
> The smaller the stripe width becomes, the more likely it is that you will need to spend time distributing the files on the volumes, and the closer the procedure becomes to manually distributing I/O.

Manually Distributing I/O

If your system does not have an LVM or hardware striping, then I/O must be manually balanced across the available disks by distributing the files according to each file's I/O requirements. In order to make decisions on file placement, you should be familiar with the I/O requirements of the database files and the capabilities of the I/O system. If you are not familiar with this data and do not have a representative workload to analyze, you can make a first guess and then tune the layout as the usage becomes known.

To stripe disks manually, you need to relate a file's storage requirements to its I/O requirements.

1. Evaluate database disk-storage requirements by checking the size of the files and the disks.
2. Identify the expected I/O throughput for each file. Determine which files have the highest I/O rate and which do not have many I/Os. Lay out the files on all the available disks so as to even out the I/O rate.

One popular approach to manual I/O distribution suggests separating a frequently used table from its index. This is not correct. During the course of a transaction, the index is read first, and then the table is read. Because these I/Os occur sequentially, the table and index can be stored on the same disk without contention. It is not sufficient to separate a data file simply because the data file contains indexes or table data. The decision to segregate a file should be made only when the I/O rate for that file affects database performance.

When to Separate Files

Regardless of whether you use operating system striping or manual I/O distribution, if the I/O system or I/O layout is not able to support the I/O rate required, then you need to separate

files with high I/O rates from the remaining files. You can identify such files either at the planning stage or after the system is live.

The decision to segregate files should only be driven by I/O rates, recoverability concerns, or manageability issues. (For example, if your LVM does not support dynamic reconfiguration of stripe width, then you might need to create smaller stripe widths to be able to add *n* disks at a time to create a new stripe of identical configuration.)

Before segregating files, verify that the bottleneck is truly an I/O issue. The data produced from investigating the bottleneck identifies which files have the highest I/O rates.

The following sections describe how to segregate the following file types:

- Tables, Indexes, and TEMP Tablespaces
- Redo Log Files
- Archived Redo Logs

Tables, Indexes, and TEMP Tablespaces

If the files with high I/O are data files belonging to tablespaces that contain tables and indexes, then identify whether the I/O for those files can be reduced by tuning SQL or application code.

If the files with high-I/O are data files that belong to the `TEMP` tablespace, then investigate whether to tune the SQL statements performing disk sorts to avoid this activity, or to tune the sorting.

After the application has been tuned to avoid unnecessary I/O, if the I/O layout is still not able to sustain the required throughput, then consider segregating the high-I/O files.

Redo Log Files

If the high-I/O files are redo log files, then consider splitting the redo log files from the other files. Possible configurations can include the following:

- Placing all redo logs on one disk without any other files. Also consider availability; members of the same group should be on different physical disks and controllers for recoverability purposes.
- Placing each redo log group on a separate disk that does not store any other files.
- Striping the redo log files across several disks, using an operating system striping tool. (Manual striping is not possible in this situation.)
- Avoiding the use of RAID 5 for redo logs.

Redo log files are written sequentially by the Log Writer (LGWR) process. This operation can be made faster if there is no concurrent activity on the same disk. Dedicating a separate disk to redo log files usually ensures that LGWR runs smoothly with no further tuning necessary. If your system supports asynchronous I/O but this feature is not currently configured, then test to see if using this feature is beneficial. Performance bottlenecks related to LGWR are rare.

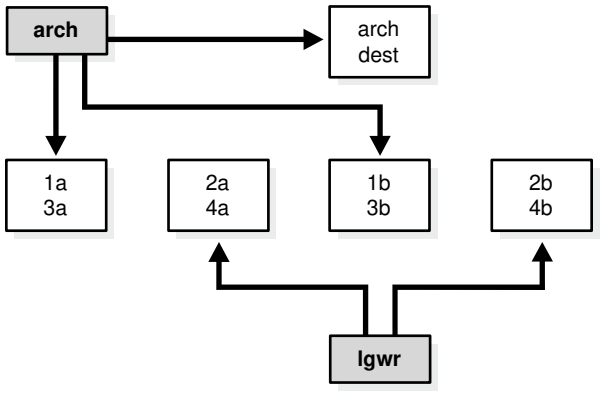
Archived Redo Logs

If the archiver is slow, then it might be prudent to prevent I/O contention between the archiver process and LGWR by ensuring that archiver reads and LGWR writes are separated. This is achieved by placing logs on alternating drives.

For example, suppose a system has four redo log groups, each group with two members. To create separate-disk access, the eight log files should be labeled 1a, 1b, 2a, 2b, 3a, 3b, 4a, and 4b. This requires at least four disks, plus one disk for archived files.

The following figure illustrates how redo members should be distributed across disks to minimize contention.

Figure 17-1 Distributing Redo Members Across Disks



In this example, LGWR switches out of log group 1 (member 1a and 1b) and writes to log group 2 (2a and 2b). Concurrently, the archiver process reads from group 1 and writes to its archive destination. Note how the redo log files are isolated from contention.

> **Note:**
>
> Mirroring redo log files, or maintaining multiple copies of each redo log file on separate disks, does not slow LGWR considerably. LGWR writes to each disk in parallel and waits until each part of the parallel write is complete. Thus, a parallel write does not take longer than the longest possible single-disk write.

Because redo logs are written serially, drives dedicated to redo log activity generally require limited head movement. This significantly accelerates log writing.

Three Sample Configurations

This section contains three high-level examples of configuring I/O systems. These examples include sample calculations that define the disk topology, stripe depths, and so on:

- Stripe Everything Across Every Disk

Stripe Everything Across Every Disk

The simplest approach to I/O configuration is to build one giant volume, striped across all available disks. To account for recoverability, the volume is mirrored (RAID 1). The striping unit for each disk should be larger than the maximum I/O size for the frequent I/O operations. This provides adequate performance for most cases.

Move Archive Logs to Different Disks

If archived redo log files are striped on the same set of disks as other files, then any I/O requests on those disks could suffer when the database is archiving the redo logs. Moving archived redo log files to separate disks provides the following benefits:

- The archive can be performed at very high rate (using sequential I/O).
- Nothing else is affected by the degraded response time on the archive destination disks.

The number of disks for archive logs is determined by the rate of archive log generation and the amount of archive storage required.

Move Redo Logs to Separate Disks

In high-update OLTP systems, the redo logs are write-intensive. Moving the redo log files to disks that are separate from other disks and from archived redo log files has the following benefits:

- Writing redo logs is performed at the highest possible rate. Hence, transaction processing performance is at its best.
- Writing of the redo logs is not impaired with any other I/O.

The number of disks for redo logs is mostly determined by the redo log size, which is generally small compared to current technology disk sizes. Typically, a configuration with two disks (possibly mirrored to four disks for fault tolerance) is adequate. In particular, by having the redo log files alternating on two disks, writing redo log information to one file does not interfere with reading a completed redo log for archiving.

Oracle Managed Files

When file systems can contain all Oracle Database data, database administration is simplified by using **Oracle Managed Files**. Oracle Database internally uses standard file system interfaces to create and delete files as needed for tablespaces, temp files, online logs, and control files. Administrators only specify the file system directory to be used for a particular type of file. You can specify one default location for data files and up to five multiplexed locations for the control and online redo log files.

Oracle Database ensures that a unique file is created and then deleted when it is no longer needed. This reduces corruption caused by administrators specifying the wrong file, reduces wasted disk space consumed by obsolete files, and simplifies creation of test and development databases. It also makes development of portable third-party

tools easier, because it eliminates the need to put operating system-specific file names in SQL scripts.

New files can be created as Oracle Managed Files, while old ones are administered in the old way. Thus, a database can have a mixture of Oracle Managed Files and user-managed files.

> **Note:**
>
> Oracle Managed Files cannot be used with raw devices.

Several points should be considered when tuning Oracle Managed Files:

- Because Oracle Managed Files require the use of a file system, DBAs give up control over how the data is laid out. Therefore, it is important to correctly configure the file system.
- Build the file system for Oracle Managed Files on top of an LVM that supports striping. For load balancing and improved throughput, stripe the disks in the file system.
- Oracle Managed Files work best if used on an LVM that supports dynamically extensible logical volumes. Otherwise, configure the logical volumes as large as possible.
- Oracle Managed Files work best if the file system provides large extensible files.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for detailed information about using Oracle Managed Files

Choosing Data Block Size

A block size of 8 KB is optimal for most systems. However, OLTP systems occasionally use smaller block sizes and DSS systems occasionally use larger block sizes. This section discusses considerations when choosing database block size for optimal performance and contains the following topics:

- Reads
- Writes
- Block Size Advantages and Disadvantages

> **Note:**
>
> The use of multiple block sizes in a single database instance is not encouraged because of manageability issues.

Reads

Regardless of the size of the data, the goal is to minimize the number of reads required to retrieve the desired data.

- If the rows are small and access is predominantly random, then choose a smaller block size.
- If the rows are small and access is predominantly sequential, then choose a larger block size.
- If the rows are small and access is both random and sequential, then it might be effective to choose a larger block size.
- If the rows are large, such as rows containing large object (LOB) data, then choose a larger block size.

Writes

For high-concurrency OLTP systems, consider appropriate values for `INITRANS`, `MAXTRANS`, and `FREELISTS` when using a larger block size. These parameters affect the degree of update concurrency allowed within a block. However, you do not need to specify the value for `FREELISTS` when using automatic segment-space management.

If you are uncertain about which block size to choose, then try a database block size of 8 KB for most systems that process a large number of transactions. This represents a good compromise and is usually effective. Only systems processing LOB data need more than 8 KB.

> **See Also:**
>
> The Oracle Database installation documentation specific to your operating system for information about the minimum and maximum block size on your platform

Block Size Advantages and Disadvantages

Table 17-3 lists the advantages and disadvantages of different block sizes.

Table 17-3 Block Size Advantages and Disadvantages

Block Size	Advantages	Disadvantages
Smaller	Good for small rows with lots of random access. Reduces block contention.	Has relatively large space overhead due to metadata (that is, block header). Not recommended for large rows. There might only be a few rows stored for each block, or worse, row chaining if a single row does not fit into a block,

Table 17-3 (Cont.) Block Size Advantages and Disadvantages

Block Size	Advantages	Disadvantages
Larger	Has lower overhead, so there is more room to store data. Permits reading several rows into the buffer cache with a single I/O (depending on row size and block size). Good for sequential access or very large rows (such as LOB data).	Wastes space in the buffer cache, if you are doing random access to small rows and have a large block size. For example, with an 8 KB block size and 50 byte row size, you waste 7,950 bytes in the buffer cache when doing random access. Not good for index blocks used in an OLTP environment, because they increase block contention on the index leaf blocks.

I/O Calibration Inside the Database

The I/O calibration feature of Oracle Database enables you to assess the performance of the storage subsystem, and determine whether I/O performance problems are caused by the database or the storage subsystem. Unlike other external I/O calibration tools that issue I/Os sequentially, the I/O calibration feature of Oracle Database issues I/Os randomly using Oracle data files to access the storage media, producing results that more closely match the actual performance of the database.

The section describes how to use the I/O calibration feature of Oracle Database and contains the following topics:

- Prerequisites for I/O Calibration
- Running I/O Calibration

Oracle Database also provides Orion, an I/O calibration tool. Orion is a tool for predicting the performance of an Oracle database without having to install Oracle or create a database. Unlike other I/O calibration tools, Oracle Orion is expressly designed for simulating Oracle database I/O workloads using the same I/O software stack as Oracle. Orion can also simulate the effect of striping performed by Oracle Automatic Storage Management. For more information, see "I/O Calibration with the Oracle Orion Calibration Tool".

Prerequisites for I/O Calibration

Before running I/O calibration, ensure that the following requirements are met:

- The user must be granted the `SYSDBA` privilege
- `timed_statistics` must be set to `TRUE`
- Asynchronous I/O must be enabled

  When using file systems, asynchronous I/O can be enabled by setting the `FILESYSTEMIO_OPTIONS` initialization parameter to `SETALL`.
- Ensure that asynchronous I/O is enabled for data files by running the following query:

```
COL NAME FORMAT A50
SELECT NAME,ASYNCH_IO FROM V$DATAFILE F,V$IOSTAT_FILE I
WHERE  F.FILE#=I.FILE_NO
AND    FILETYPE_NAME='Data File';
```

Additionally, only one calibration can be performed on a database instance at a time.

Running I/O Calibration

The I/O calibration feature of Oracle Database is accessed using the `DBMS_RESOURCE_MANAGER.CALIBRATE_IO` procedure. This procedure issues an I/O intensive read-only workload, made up of one megabyte of random of I/Os, to the database files to determine the maximum IOPS (I/O requests per second) and MBPS (megabytes of I/O per second) that can be sustained by the storage subsystem.

The I/O calibration occurs in two steps:

- In the first step of I/O calibration with the `DBMS_RESOURCE_MANAGER.CALIBRATE_IO` procedure, the procedure issues random database-block-sized reads, by default, 8 KB, to all data files from all database instances. This step provides the maximum IOPS, in the output parameter `max_iops`, that the database can sustain. The value `max_iops` is an important metric for OLTP databases. The output parameter `actual_latency` provides the average latency for this workload. When you need a specific target latency, you can specify the target latency with the input parameter `max_latency` (specifies the maximum tolerable latency in milliseconds for database-block-sized IO requests).
- The second step of calibration using the `DBMS_RESOURCE_MANAGER.CALIBRATE_IO` procedure issues random, 1 MB reads to all data files from all database instances. The second step yields the output parameter `max_mbps`, which specifies the maximum MBPS of I/O that the database can sustain. This step provides an important metric for data warehouses.

The calibration runs more efficiently if the user provides the `num_physical_disks` input parameter, which specifies the approximate number of physical disks in the database storage system.

Due to the overhead from running the I/O workload, I/O calibration should only be performed when the database is idle, or during off-peak hours, to minimize the impact of the I/O workload on the normal database workload.

To run I/O calibration and assess the I/O capability of the storage subsystem used by Oracle Database, use the `DBMS_RESOURCE_MANAGER.CALIBRATE_IO` procedure:

```
SET SERVEROUTPUT ON
DECLARE
  lat  NUMBER;
  iops INTEGER;
  mbps INTEGER;
BEGIN
-- DBMS_RESOURCE_MANAGER.CALIBRATE_IO (<DISKS>, <MAX_LATENCY>, iops, mbps, lat);
   DBMS_RESOURCE_MANAGER.CALIBRATE_IO (2, 10, iops, mbps, lat);

end;
/
```

When running the `DBMS_RESOURCE_MANAGER.CALIBRATE_IO` procedure, consider the following:

- Only run one calibration at a time on databases that use the same storage subsystem. If you simultaneously run the calibration across separate databases that use the same storage subsystem, the calibration will fail.
- Quiesce the database to minimize I/O on the instance.

- For Oracle Real Application Clusters (Oracle RAC) configurations, ensure that all instances are opened to calibrate the storage subsystem across nodes.
- For an Oracle Real Application Clusters (Oracle RAC) database, the workload is simultaneously generated from all instances.
- The `num_physical_disks` input parameter is optional. By setting the `num_physical_disks` parameter to the approximate number of physical disks in the database's storage system, the calibration can be faster and more accurate.
- In some cases, asynchronous I/O is permitted for data files, but the I/O subsystem for submitting asynchronous I/O may be maximized, and I/O calibration cannot continue. In such cases, refer to the port-specific documentation for information about checking the maximum limit for asynchronous I/O on the system.

At any time during the I/O calibration process, you can query the calibration status in the `V$IO_CALIBRATION_STATUS` view. After I/O calibration is successfully completed, you can view the results in the `DBA_RSRC_IO_CALIBRATE` table.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about running the `DBMS_RESOURCE_MANAGER.CALIBRATE_IO` procedure
> - *Oracle Database Reference* for more information about the `V$IO_CALIBRATION_STATUS` view and `DBA_RSRC_IO_CALIBRATE` table

I/O Calibration with the Oracle Orion Calibration Tool

This section describes the Oracle Orion Calibration Tool and includes the following sections:

Introduction to the Oracle Orion Calibration Tool

Oracle Orion is a tool for predicting the performance of an Oracle database without having to install Oracle or create a database. Unlike other I/O calibration tools, Oracle Orion is expressly designed for simulating Oracle database I/O workloads using the same I/O software stack as Oracle. Orion can also simulate the effect of striping performed by Oracle Automatic Storage Management.

Table 17-4 lists the types of I/O workloads that Orion supports.

For each type of workload shown in Table 17-4, Orion can run tests using different I/O loads to measure performance metrics such as MBPS, IOPS, and I/O latency. Load is expressed in terms of the number of outstanding asynchronous I/Os. Internally, for each such load level, the Orion software keeps issuing I/O requests as fast as they complete to maintain the I/O

load at that level. For random workloads, using either large or small sized I/Os, the load level is the number of outstanding I/Os. For large sequential workloads, the load level is a combination of the number of sequential streams and the number of outstanding I/Os per stream. Testing a given workload at a range of load levels can help you understand how performance is affected by load.

Note the following when you use Orion:

- Run Orion when the storage is idle (or pretty close to idle). Orion calibrates the performance of the storage based on the I/O load it generates; Orion is not able to properly assess the performance if non-Orion I/O workloads run simultaneously.
- If a database has been created on the storage, the storage can alternatively be calibrated using the PL/SQL routine `dbms_resource_manager.calibrate_io()`.

Table 17-4 Orion I/O Workload Support

Workload	Description
Small Random I/O	OLTP applications typically generate random reads and writes whose size is equivalent to the database block size, typically 8 KB. Such applications typically care about the throughput in I/Os Per Second (IOPS) and about the average latency (I/O turn-around time) per request. These parameters translate to the transaction rate and transaction turn-around time at the application layer. Orion simulates a random I/O workload with a given percentage of reads compared to writes, a given I/O size, and a given number of outstanding I/Os. In this Orion workload simulation, the I/Os are distributed across all disks.
Large Sequential I/O	Data warehousing applications, data loads, backups, and restores generate sequential read and write streams composed of multiple outstanding 1 MB I/Os. Such applications are processing large amounts of data, such as a whole table or a whole database and they typically care about the overall data throughput in MegaBytes Per Second (MBPS). Orion can simulate a given number of sequential read or write streams of a given I/O size with a given number of outstanding I/Os. Orion can optionally simulate Oracle Automatic Storage Management striping when testing sequential streams.
Large Random I/O	A sequential stream typically accesses the disks concurrently with other database traffic. With striping, a sequential stream is spread across many disks. Consequently, at the disk level, multiple sequential streams are seen as random 1 MB I/Os.
Mixed Workloads	Orion can simulate two simultaneous workloads: Small Random I/O and either Large Sequential I/O or Large Random I/O. This workload type enables you to simulate, for example, an OLTP workload of 8 KB random reads and writes with a backup workload of four sequential read streams of 1 MB I/Os.

Each Orion data point is a test for a specific mix of small and large I/O loads sustained for a duration. An Orion test consists of multiple data point tests. These data point tests can be represented as a two-dimensional matrix. Each column in the matrix represents data point tests with the same small I/O load, but varying large I/O loads. Each row represents data point tests with the same large I/O load, but varying small I/O loads. An Orion test can be for a single point, a single row, a single column, or for the whole matrix.

Orion Test Targets

You can use Orion to test any disk-based character device that supports asynchronous I/O. Orion has been tested on the following types of targets:

- DAS (direct-attached) storage: You can use Orion to test the performance of one or more local disks, volumes, or files on the local host.
- SAN (storage-area network) storage: Orion can be run on any host that has all or parts of the SAN storage mapped as character devices. The devices can correspond to striped or un-striped volumes exported by the storage array(s), or individual disks, or one or more whole arrays.
- NAS (network-attached storage): You can use Orion to test the performance on data files on NAS storage. In general, the performance results on NAS storage are dependent on the I/O patterns with which the data files have been created and updated. Therefore, you should initialize the data files appropriately before running Orion.

Orion for Oracle Administrators

Oracle administrators can use Orion to evaluate and compare different storage arrays, based on the expected workloads. Oracle administrators can also use Orion to determine the optimal number of network connections, storage arrays, storage array controllers, and disks for the expected peak workloads.

Getting Started with Orion

To get started using Orion, do the following:

1. Select a test name to use with the Orion `-testname` parameter. This parameter specifies a unique identifier for your Orion run. For example, use the test name "mytest". For more information, see "Orion Parameters".
2. Create an Orion input file, based on the test name. For example, create a file named `mytest.lun`. In the input file list the raw volumes or files to test. Add one volume name per line. Do not put comments or anything else in the `.lun` file.

   For example, an Orion input file could contain the following:

   ```
   /dev/raw/raw1
   /dev/raw/raw2
   /dev/raw/raw3
   /dev/raw/raw4
   /dev/raw/raw5
   /dev/raw/raw6
   /dev/raw/raw7
   /dev/raw/raw8
   ```

   For more information, see "Orion Input Files".
3. Verify that the all volumes specified in the input file, for example mytest.lun, are accessible using the command `dd` or another equivalent file viewing utility. For example, for a typical sanity-check try the following on a Linux system:

   ```
   $ dd if=/dev/raw/raw1 of=/dev/null bs=32k count=1024
   ```

   Depending on your platform, the file viewing utility you use and its interface may be different.
4. Verify that your platform has the necessary libraries installed to do asynchronous I/Os. The Orion test is completely dependent on asynchronous I/O. On Linux and Solaris, the library `libaio` must be in the standard `lib` directories or accessible through the shell environment's library path variable (usually `LD_LIBRARY_PATH` or `LIBPATH`, depending on your shell). Windows has built-in asynchronous I/O libraries, so this issue does not apply.

5. As a first test with Orion, use `-run` with either the `oltp` or `dss` option. If the database is primarily OLTP, then use `-run oltp`. If the database is primarily for data warehousing or analytics, then use `-run dss`.

   For example, use the following command to run an OLTP-like workload using the default input file name, `orion.lun`:

   ```
   $ ./orion -run oltp
   ```

   The I/O load levels generated by Orion take into account the number of disk spindles being tested (or specified with the `-num_disks` parameter). Keep in mind that the number of spindles *may or may not be* related to the number of volumes specified in the input file, depending on how these volumes are mapped.

6. The section "Orion Output Files" provides sample results showing the Orion output files. Using the sample file `mytest_summary.txt` is a good starting point for verifying the input parameters and analyzing the output. The sample files `mytest_*.csv` contain comma-delimited values for several I/O performance measures.

Orion Input Files

When you specify the Orion `-testname` <*testname*> parameter, this sets the test name prefix for the Orion input and output filenames. The default value for the `-testname` option is "orion".

The Orion input file, <*testname*>.`lun` should contain a carriage-return-separated list of LUNs.

Orion Parameters

Use the Orion command parameters to specify the I/O workload type and to specify other Orion options.

Orion Required Parameter

The `-run` parameter is required with the Orion command. Table 17-5 describes the `-run` parameter.

Table 17-5 Required Orion Parameter

Option	Description	Default

`-run` *level*	Specifies the test run level to be *level*. This option provides the run level and allows complex commands to be specified at the advanced level. If not set as `-run advanced`, then setting any other parameter, besides `-cache_size` or `-verbose`, results in an error.
	Except **advanced**, all of the `-run` *level* settings use a pre-specified set of parameters.
	The *level* must be one of:
	• **oltp**
	Tests with random small (8K) I/Os at increasing loads to determine the maximum IOPS.
	This parameter corresponds to the following Orion invocation:
	`%> ./orion -run advanced \`
	`-num_large 0 -size_small 8 -type rand \`
	`-simulate concat -write 0 -duration 60 \`
	`-matrix row`
	• **dss**
	Tests with random large (1M) I/Os at increasing loads to determine the maximum throughput.
	This parameter corresponds to the following Orion invocation:
	`%> ./orion -run advanced \`
	`-num_small 0 -size_large 1024 -type rand \`
	`-simulate concat -write 0 -duration 60 \`
	`-matrix column`
	• **simple**
	Generates the Small Random I/O and the Large Random I/O workloads for a range of load levels. In this option, small and large I/Os are tested in isolation. The only optional parameters that can be specified at this run level are `-cache_size` and `-verbose`.
	This parameter corresponds to the following Orion invocation:
	`%> ./orion -run advanced \`
	`-size_small 8 -size_large 1024 -type rand \`
	`-simulate concat -write 0 -duration 60 \`
	`-matrix basic`
	• **normal**
	Same as `simple`, but also generates combinations of the small random I/O and large random I/O workloads for a range of loads. The only optional parameters that can be specified at this run level are `-cache_size` and `-verbose`.
	This parameter corresponds to the following Orion invocation:
	`%> ./orion -run advanced \`
	`-size_small 8 -size_large 1024 -type rand \`
	`-simulate concat -write 0 -duration 60 \`
	`-matrix detailed`
	• **advanced**
	Tests the workload you specify with optional parameters. Any of the optional parameters can be specified at this run level.
		`normal`

Orion Optional Parameters

Table 17-6 Optional Orion Parameters

Option	Description	Default
`-cache_size` *num*	Size of the storage array's read or write cache (in MB). For Large Sequential I/O workloads, Orion warms the cache by doing random large I/Os before each data point. Orion uses the cache size to determine the duration for this cache warming operation. If set to 0, do not perform cache warming. Unless this option is set to 0, Orion issues several unmeasured, random I/Os before each large sequential data point. These I/Os fill up the storage array's cache, if any, with random data so that I/Os from one data point do not result in cache hits for the next data point. Read tests are preceded with junk reads and write tests are preceded with junk writes. If specified, this 'cache warming' is performed until *num* MBs of I/O have been read or written.	Default Value: If not specified, warming occurs for a default amount of time (two minutes). That is, issue two minutes of unmeasured random I/Os before each data point.
`-duration` *num_seconds*	Set the duration to test each data point in seconds to the value *num_seconds*.	Default Value: `60`
`-help`	Prints Orion help information. All other options are ignored with help set.	
`-matrix` *type*	Type of mixed workloads to test over a range of loads. An Orion test consists of multiple data point tests. The data point tests can be represented as a two-dimensional matrix. Each column in the matrix represents data point tests with the same small I/O load, but varying large I/O loads. Each row represents data point tests with the same large I/O load, but varying small I/O loads. An Orion test can be for a single point, a single row, a single column, or the whole matrix, depending on the matrix *type*: • **basic:** No mixed workload. The Small Random and Large Random/Sequential workloads are tested separately. Test small I/Os only, then large I/Os only. • **detailed:** Small Random and Large Random/Sequential workloads are tested in combination. Test entire matrix. • **point:** A single data point with *S* outstanding Small Random I/Os and *L* outstanding Large Random I/Os or sequential streams. *S* is set by the `-num_small` parameter. *L* is set by the `-num_large` parameter. Test with `-num_small` small I/Os, `-num_large` large I/Os. • **col:** Large Random/Sequential workloads only. Test a varying large I/O load with `-num_small` small I/Os. • **row:** Small Random workloads only. Test a varying small I/O load with `-num_large` large I/Os. • **max:** Same as detailed, but only tests the workload at the maximum load, specified by the `-num_small` and `-num_large` parameters. Test varying loads up to the `-num_small` and `-num_large` limits.	Default Value: `basic`

Table 17-6 (Cont.) Optional Orion Parameters

Option	Description	Default
–num_disks *value*	Specify the number of physical disks used by the test. Used to generate a range for the load. Specifies the number of disks (physical spindles). This number *value* is used to gauge the range of loads that Orion should test at. Increasing this parameter results in Orion using heavier I/O loads.	Default Value: the number of LUNs in <*testname*>.lun.
-num_large *value*	Controls the large I/O load. Note, this option only applies when -matrix is specified as: row, point, or max. When the -type option is set to rand, the parameter argument *value* specifies the number of outstanding large I/Os. When the -type option is set to seq, the parameter argument *value* specifies the number of sequential I/O streams.	Default Value: no default
-num_small	Specify the maximum number of outstanding I/Os for the Small Random I/O workload. Note: this only applies when -matrix is specified as col, point, or max.	Default Value: no default
–num_streamIO *num*	Specify the number of concurrent I/Os per stream as *num*. Note: this parameter is only used if -type is seq.	Default Value: 4
-simulate *type*	Data layout to simulate for Large Sequential I/O workload. Orion tests on a virtual LUN formed by combining specified LUNs in one of these ways. The *type* is one: • **concat:** A virtual volume is simulated by serially chaining the specified LUNs. A sequential test over this virtual volume will go from some point to the end of each one LUN, followed by the beginning to end of the next LUN, and so on. • **raid0:** A virtual volume is simulated by striping across the specified LUNs. Each sequential stream issues I/Os across all LUNs using raid0 striping. The stripe depth is 1M by default, to match the Oracle Automatic Storage Management stripe depth, and can be changed with the -stripe parameter. The offsets for I/Os are determined as follows: For Small Random and Large Random workloads: • The LUNs are concatenated into a single virtual LUN (VLUN) and random offsets are chosen within the VLUN. For Large Sequential workloads: • With striping (-simulate raid0). The LUNs are used to create a single striped VLUN. With no concurrent Small Random workload, the sequential streams start at fixed offsets within the striped VLUN. For *n* streams, stream *i* start at offset VLUNsize * (*i* + 1) / (*n* + 1), unless *n* is 1, in which case the single stream start at offset 0. With a concurrent Small Random workload, streams start at random offsets within the striped VLUN. • Without striping (-simulate CONCAT). The LUNs are concatenated into a single VLUN. The streams start at random offsets within the single VLUN. This parameter is typically only used if -type is seq.	Default Value: concat

Table 17-6 (Cont.) Optional Orion Parameters

Option	Description	Default
`-size_large` *num*	Specify the *num*, size of the I/Os (in KB) for the Large Random or Sequential I/O workload.	Default Value: `1024`
`-size_small` *num*	Specify the num, size of the I/Os (in KB) for the Small Random I/O workload.	Default Value: `8`
`-storax` *type*	API to use for testing I/O workload. • **skgfr:** Use operating system I/O layer. • **oss:** Use OSS API for I/O with Cell server in an Exadata machine. • **asmlib:** Use ASMLIB disk devices based storage API for I/O. • **odmlib:** Use Direct NFS storage based API for I/O.	Default Value: `skgfr`
`-testname` *tname*	Specify the *tname* identifier for the test run. When specified, the input file containing the LUN disk or file names must be named <*tname*>.lun. The output files are named with the prefix <*tname*>_.	Default Value: `orion`
`-type` [`rand` | `seq`]	Type of the Large I/O workload. • **rand**: Randomly distributed large I/Os. • **seq**: Sequential streams of large I/Os.	Default Value: `rand`
`-verbose`	Prints status and tracing information to standard output.	Default Value: option not set
`-write` *num_write*	Specify the percentage of I/Os that are writes to *num_write*; the rest being reads. This parameter applies to both the Large and Small I/O workloads. For Large Sequential I/Os, each stream is either read-only or write-only; the parameter specifies the percentage of streams that are write-only. The data written to disk is garbage and unrelated to any existing data on the disk. **Caution**: write tests obliterate all data on the specified LUNS.	Default Value: `0`

> **Note:**
>
> Write tests obliterate all data on the specified LUNS.

Orion Command Line Samples

The following provides sample Orion commands for different types of I/O workloads:

1. To evaluate storage for an OLTP database:

   ```
   -run oltp
   ```

2. To evaluate storage for a data warehouse:

   ```
   -run dss
   ```

3. For a basic set of data:

   ```
   -run normal
   ```

4. To understand your storage performance with read-only, small and large random I/O workload:

```
$ orion -run simple
```

5. To understand your storage performance with a mixed small and large random I/O workload:

```
$ orion -run normal
```

6. To generate combinations of 32KB and 1MB reads to random locations:

```
$ orion -run advanced -size_small 32 \
-size_large 1024 -type rand -matrix detailed
```

7. To generate multiple sequential 1 MB write streams, simulating 1 MB RAID-0 stripes:

```
$ orion -run advanced -simulate raid0 \
-stripe 1024 -write 100 -type seq -matrix col -num_small 0
```

8. To generate combinations of 32 KB and 1 MB reads to random locations:

```
 -run advanced -size_small 32 -size_large 1024 -type rand -matrix detailed
```

9. To generate multiple sequential 1 MB write streams, simulating RAID0 striping:

```
 -run advanced -simulate raid0 -write 100 -type seq -matrix col -num_small 0
```

Orion Output Files

The output files for a test run are prefixed by <*testname*>_<*date*> where *date* is *yyyymmdd_hhmm*.

Table 17-7 lists the Orion output files.

Table 17-7 Orion Generated Output Files

Output File	Description
<*testname*>_<*date*>_hist.csv	Histogram of I/O latencies.
<*testname*>_<*date*>_iops.csv	Performance results of small I/Os in IOPS.
<*testname*>_<*date*>_lat.csv	Latency of small I/Os in microseconds.
<*testname*>_<*date*>_mbps.csv	Performance results of large I/Os in MBPS.
<*testname*>_<*date*>_summary.txt	Summary of the input parameters, along with the minimum small I/O latency (in secs), the maximum MBPS, and the maximum IOPS observed.
<*testname*>_<*date*>_trace.txt	Extended, unprocessed output.

> **Note:**
>
> If you are performing write tests, be prepared to lose any data stored on the LUNs.

Orion Sample Output Files

Orion creates several output files as specified in Table 17-7. For the sample "mytest" shown in the section, "Getting Started with Orion", the output files are:

- **mytest_summary.txt**: This file contains:
  - Input parameters
  - Maximum throughput observed for the Large Random/Sequential workload
  - Maximum I/O rate observed for the Small Random workload
  - Minimum latency observed for the Small Random workload
- **mytest_mbps.csv:** comma-delimited value file containing the data transfer rate (MBPS) results for the Large Random/Sequential workload. In the general case, this and all other CSV files contains a two-dimensional table. Each row in the table corresponds to a large I/O load level and each column corresponds to a specific small I/O load level. Thus, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (for random large I/O tests) or the number of sequential streams (for sequential large I/O tests).

  The following example shows the first few data points of the Orion MBPS output CSV file for "mytest". The simple mytest command-line does not test combinations of large and small I/Os. Hence, the MBPS file has just one column corresponding to 0 outstanding small I/Os. In this example, at a load level of 8 outstanding large reads and no small I/Os, the report data indicates a throughput of 103.06 MBPS.

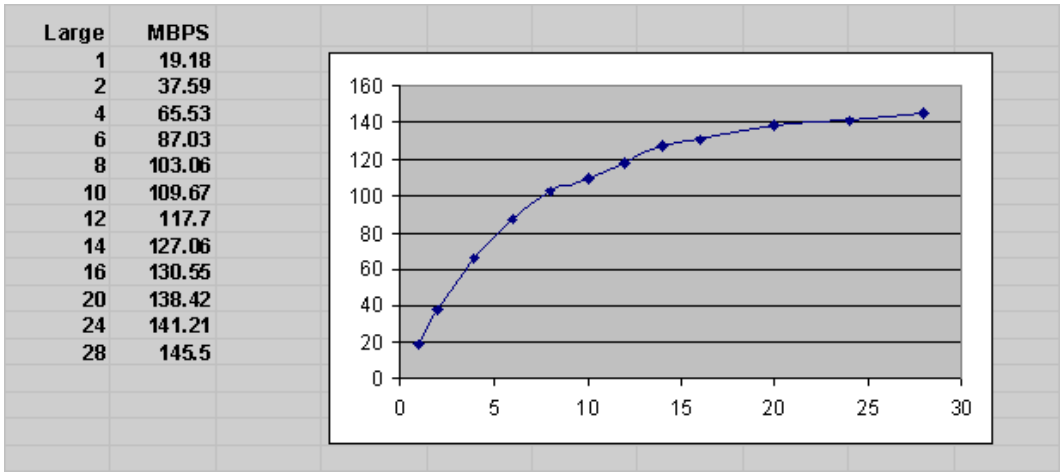
```
Large/Small,       0

1,   19.18
2,   37.59
4,   65.53
6,   87.03
8,  103.06
10,  109.67
. .   . . .
. .   . . .
```

  The following graph shows a sample data transfer rate measured at different large I/O load levels. This graph can be generated by loading `mytest_mbps.csv` into a spreadsheet and graphing the data points. Orion does not directly generate such graphs. The x-axis corresponds to the number of outstanding large reads and the y-axis corresponds to the throughput observed.

  The graph shows typical storage system behavior. As the number of outstanding I/O requests is increased, the throughput increases. However, at a certain point the throughput level stabilizes, indicating the storage system's maximum throughput value.

Figure 17-2 Sample I/O Load Levels



- **mytest_iops.csv**: Comma-delimited value file containing the I/O throughput (in IOPS) results for the Small Random workload. Like in the MBPS file, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os, when testing large random, or the number of sequential streams (for large sequential).

  In the general case, a CSV file contains a two-dimensional table. However, for a simple test where you are not testing combinations of large and small I/Os the results file has just one row. Hence, the IOPS results file just has one row with 0 large I/Os. As shown in the following example, data point with 12 outstanding small reads and no large I/Os provides a sample throughput of 951 IOPS.

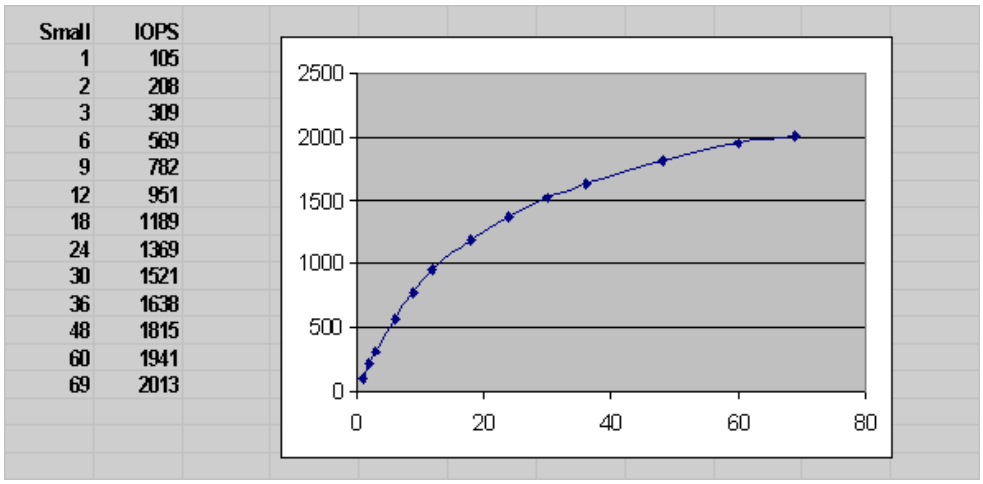
```
Large/Small,      1,      2,      3,      6,      9,     12 . . . .

0,              105,    208,    309,    569,    782,    951 . . . .
```

  The following graph is generated by loading `mytest_iops.csv` into Excel and charting the data. This graph illustrates the IOPS throughput seen at different small I/O load levels.

  The graph shows typical storage system behavior. As the number of outstanding I/O requests is increased, the throughput increases. However, at a certain point, the throughput level stabilizes, indicating the storage system reaches a maximum throughput value. At higher throughput levels, the latency for the I/O requests also increase significantly. Therefore, it is important to view this data with the latency data provided in the generated latency results in mytest_lat.csv.

Figure 17-3 I/O Throughput at Different Small I/O Load Levels



- **mytest_lat.csv**: Comma-delimited value file containing the latency results for the Small Random workload. As with the MBPS and IOPS files, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (when testing large random I/Os) or the number of sequential streams.

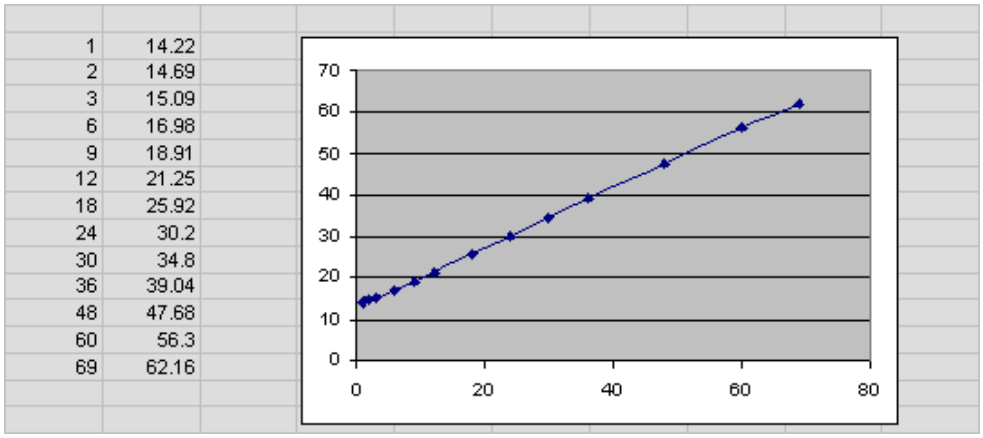
  In the general case, a CSV file contains a two-dimensional table. However, for a simple test where you are not testing combinations of large and small I/Os the results file has just one row. Hence, the IOPS results file just has one row with 0 large I/Os. In the following example, at a sustained load level of 12 outstanding small reads and no large I/Os, the generated results show an I/O turn-around latency of 22.25 milliseconds.

  ```
  Large/Small, 1, 2, 3, 6, 9, 12 . . . .

  0, 14.22, 14.69, 15.09, 16.98, 18.91, 21.25 . . . .
  ```

  The following graph is generated by loading `mytest_lat.csv` into Excel and charting the data. This graph illustrates the small I/O latency at different small I/O load levels for mytest.

Figure 17-4 I/O Latency at Small I/O Load Levels



- **mytest_trace.txt**: Contains the extended, unprocessed test output.

> **Note:**
>
> Orion reports errors that occur during a test on standard output.

Orion Troubleshooting

1. If you are getting an I/O error on one or more of the volumes specified in the <*testname*>.lun file:
   - Verify that you can access the volume in the same mode as the test, read or write, using a file copy program such as `dd`.
   - Verify that your host operating system version can do asynchronous I/O.
   - On Linux and Solaris, the library `libaio` must be in the standard lib directories or accessible through the shell environment's library path variable (usually LD_LIBRARY_PATH or LIBPATH, depending on your shell).
2. If you run on NAS storage:
   - The file system must be properly mounted for Orion to run. Please consult your Oracle Installation Guide for directions (for example, the section, Appendix B "Using NAS Devices" in the Database Installation Guide for Linux x86).
   - The mytest.lun file should contain one or more paths of existing files. Orion does not work on directories or mount points. The file has to be large enough for a meaningful test. The size of this file should represent the eventual expected size of your datafiles (say, after a few years of use).
   - You may see poor performance doing asynchronous I/O over NFS on Linux (including 2.6 kernels).
   - If you are doing read tests and the reads are hitting untouched blocks of the file that were not initialized or previously written, some smart NAS systems may "fake" the read by returning zeroed-out blocks. When this occurs, you see unexpectedly good performance.

     The workaround is to write all blocks, using a tool such as `dd`, before performing the read test.
3. If you run Orion on Windows: Testing on raw partitions requires temporarily mapping the partitions to drive letters and specifying these drive letters in the test.lun file.
4. If you run Orion 32-bit Linux/x86 binary on an x86_64 system: Please copy a 32-bit libaio.so file from a 32-bit computer running the same Linux version.
5. If you are testing with a lot of disks (`num_disks` greater than around 30):
   - You should use the -duration option (see the optional parameters section for more details) to specify a long duration (like 120 seconds or more) for each data point. Since Orion tries to keep all the spindles running at a particular load level, each data point requires a ramp-up time, which implies a longer duration for the test.
   - You may get the following error message, instructing you to increase the duration value:

     ```
     Specify a longer -duration value.
     ```

A duration of 2x the number of spindles seems to be a good rule of thumb. Depending on your disk technology, your platform may need more or less time.

6. If you get an error about libraries being used by Orion:
   - Linux/Solaris: See I/O error troubleshooting.
   - **NT-Only:** Do not move/remove the Oracle libraries included in the distribution. These must be in the same directory as orion.exe.
7. If you are seeing performance numbers that are "unbelievably good":
   - You may have a large read or write cache, or read and write cache somewhere between the Orion program and the disk spindles. Typically, the storage array controller has the biggest effect. Find out the size of this cache and use the -cache_size advanced option to specify it to Orion (see the optional parameters section for more details).
   - The total size of your volumes may be really small compared to one or more caches along the way. Try to turn off the cache. This is needed if the other volumes sharing your storage show significant I/O activity in a production environment (and end up using large parts of the shared cache).
8. If Orion is reporting a long estimated run time:
   - The run time increases when `-num_disks` is high. Orion internally uses a linear formula to determine how long it takes to saturate the given number of disks.
   - The `-cache_size` parameter affects the run time, even when it is not specified. Orion does cache warming for two minutes per data point by default. If you have turned off the cache, specify `-cache_size 0`.
   - The run time increases when a long -duration value is specified, as expected.

18
Managing Operating System Resources

This chapter explains how to tune the operating system for optimal performance of Oracle Database.

This chapter contains the following sections:

- Understanding Operating System Performance Issues
- Resolving Operating System Issues
- Understanding CPU
- Resolving CPU Issues

> **See Also:**
>
> - Your operating system documentation
> - Your Oracle Database platform-specific documentation, which contains tuning information specific to your platform

Understanding Operating System Performance Issues

Operating system performance issues commonly involve process management, memory management, and scheduling. If you have tuned the Oracle database instance and still need to improve performance, verify your work or try to reduce system time. Ensure that there is enough I/O bandwidth, CPU power, and swap space. Do not expect, however, that further tuning of the operating system will have a significant effect on application performance. Changes in the Oracle Database configuration or in the application are likely to result in a more significant difference in operating system efficiency than simply tuning the operating system.

For example, if an application experiences excessive buffer busy waits, then the number of system calls increases. If you reduce the buffer busy waits by tuning the application, then the number of system calls decreases.

This section covers the following topics related to operating system performance issues:

- Using Operating System Caches
- Memory Usage
- Using Operating System Resource Managers

Using Operating System Caches

Operating systems and device controllers provide data caches that do not directly conflict with Oracle Database cache management. Nonetheless, these structures can consume resources while offering little or no performance benefit. This situation is most noticeable

when database files are stored in a Linux or UNIX file system. By default, all database I/O goes through the file system cache.

On some Linux and UNIX systems, direct I/O is available to the filestore. This arrangement allows the database files to be accessed within the file system, bypassing the file system cache. Direct I/O saves CPU resources and allows the file system cache to be dedicated to non-database activity, such as program texts and spool files.

> **Note:**
>
> This problem does not occur on Windows. All file requests by the database bypass the caches in the file system.

Although the operating system cache is often redundant because the Oracle Database buffer cache buffers blocks, in some cases the database does not use the database buffer cache. In these cases, using direct I/O or raw devices may yield worse performance than using operating system buffering. Examples include:

- Reads or writes to the `TEMP` tablespace
- Data stored in `NOCACHE` LOBs
- Parallel execution servers reading data

> **Note:**
>
> In some cases the database can cache parallel query data in the database buffer cache instead of performing direct reads from disk into the PGA. This configuration may be appropriate when the database servers have a large amount of memory. See *Oracle Database VLDB and Partitioning Guide* to learn more using parallel execution.

You may want to cache but not all files at the operating system level.

Asynchronous I/O

With synchronous I/O, when an I/O request is submitted to the operating system, the writing process blocks until the write is confirmed as complete. It can then continue processing. With asynchronous I/O, processing continues while the I/O request is submitted and processed. Use asynchronous I/O when possible to avoid bottlenecks.

Some platforms support asynchronous I/O by default, others need special configuration, and some only support asynchronous I/O for certain underlying file system types.

FILESYSTEMIO_OPTIONS Initialization Parameter

You can use the `FILESYSTEMIO_OPTIONS` initialization parameter to enable or disable asynchronous I/O or direct I/O on file system files. This parameter is platform-specific and has a default value that is best for a particular platform.

FILESYSTEMIO_OPTIONS can be set to one of the following values:

- ASYNCH: enable asynchronous I/O on file system files, which has no timing requirement for transmission.
- DIRECTIO: enable direct I/O on file system files, which bypasses the buffer cache.
- SETALL: enable both asynchronous and direct I/O on file system files.
- NONE: disable both asynchronous and direct I/O on file system files.

> **See Also:**
>
> Your platform-specific documentation for more details

Limiting Asynchronous I/O in NFS Server Environments

In some Network File Storage (NFS) server environments, performance may be impaired if a large number of asynchronous I/O requests are made within a short period of time. In such cases, use the DNFS_BATCH_SIZE initialization parameter to improve performance and increase stability on your system by limiting the number of I/Os issued by an Oracle process.

The DNFS_BATCH_SIZE initialization parameter controls the number of asynchronous I/Os that can be queued by an Oracle foreground process when Direct NFS Client is enabled. In environments where the NFS server cannot handle a large number of outstanding asynchronous I/O requests, Oracle recommends setting this parameter to a value of 128. You can then increase or decrease its value based on the performance of your NFS server.

> **Note:**
>
> The default setting for the DNFS_BATCH_SIZE initialization parameter is 4096. The recommended value of 128 is only applicable on systems where the NFS server cannot handle a large number of asynchronous I/O requests and severe latency is detected.

> **See Also:**
>
> *Oracle Database Reference* for information about the DNFS_BATCH_SIZE initialization parameter

Improving I/O Performance Using Direct NFS Client

Direct NFS Client integrates the NFS client functionality directly in Oracle Database. Because Direct NFS Client is a specialized NFS client for Oracle Database, it is highly optimized. Direct NFS Client considerably improves database performance over NFS as compared to the traditional operating system NFS client.

Parallel NFS is an optional feature of Direct NFS Client that is introduced in NFS version 4.1 and is supported by Oracle Database 12*c* Release 2 (12.2) and later. Parallel NFS is a highly

scalable distributed storage protocol, where clients, server, and storage devices are responsible for managing file access. In the NFS versions earlier to 4.1, only the server is responsible for managing file access. Thus, Parallel NFS enables highly scalable distributed NAS storage for better I/O performance.

Starting with Oracle Database 12*c* Release 2 (12.2), you can also use the Direct NFS dispatcher feature of Direct NFS Client. The Direct NFS dispatcher consolidates the TCP connections that are created from a database instance to an NFS server. In large database deployments, using Direct NFS dispatcher improves scalability and network performance. Therefore, for a large number of TCP connections, Oracle recommends using Direct NFS dispatcher along with Parallel NFS for a Direct NFS Client deployment.

See Also:

- *Oracle Database Installation Guide* for information about enabling the Parallel NFS feature for Direct NFS Client by setting the value for the `nfs_version` parameter to `pNFS` in the Direct NFS configuration file `oranfstab`.
- *Oracle Database Reference* for information about enabling the Direct NFS dispatcher feature for the Direct NFS Client by setting the value for the `ENABLE_DNFS_DISPATCHER` initialization parameter to `true`.

Memory Usage

Memory usage is affected by both buffer cache limits and initialization parameters.

Buffer Cache Limits

The UNIX buffer cache consumes operating system memory resources. Although in some versions of UNIX, the UNIX buffer cache may be allocated a set amount of memory, it is common today for more sophisticated memory management mechanisms to be used. Typically, these will allow free memory pages to be used to cache I/O. In such systems, it is common for operating system reporting tools to show that there is no free memory, which is not generally a problem. If processes require more memory, the memory caching I/O data is usually released to allow the process memory to be allocated.

Parameters Affecting Memory Usage

The memory required by any one Oracle Database session depends on many factors. Typically the major contributing factors are:

- Number of open cursors
- Memory used by PL/SQL, such as PL/SQL tables
- `SORT_AREA_SIZE` initialization parameter

In Oracle Database, the `PGA_AGGREGATE_TARGET` initialization parameter gives greater control over a session's memory usage.

Using Operating System Resource Managers

Some platforms provide operating system resource managers. These are designed to reduce the impact of peak load use patterns by prioritizing access to system resources. They usually implement administrative policies that govern which resources users can access and how much of those resources each user is permitted to consume.

Operating system resource managers are different from domains or other similar facilities. Domains provide one or more completely separated environments within one system. Disk, CPU, memory, and all other resources are dedicated to each domain and cannot be accessed from any other domain. Other similar facilities completely separate just a portion of system resources into different areas, usually separate CPU or memory areas. Like domains, the separate resource areas are dedicated only to the processing assigned to that area; processes cannot migrate across boundaries. Unlike domains, all other resources (usually disk) are accessed by all partitions on a system.

Oracle Database runs within domains, and within these other less complete partitioning constructs, as long as the allocation of partitioned memory (RAM) resources is fixed, not dynamic.

Operating system resource managers prioritize resource allocation within a global pool of resources, usually a domain or an entire system. Processes are assigned to groups, which are in turn assigned resources anywhere within the resource pool.

Note:

- If you have multiple instances on a node, and you want to distribute resources among them, then each instance should be assigned to a dedicated operating-system resource manager group or managed entity. To run multiple instances in the managed entity, use instance caging to manage how the CPU resources within the managed entity should be distributed among the instances. When Oracle Database Resource Manager is managing CPU resources, it expects a fixed amount of CPU resources for the instance. Without instance caging, it expects the available CPU resources to be equal to the number of CPUs in the managed entity. With instance caging, it expects the available CPU resources to be equal to the value of the `CPU_COUNT` initialization parameter. If there are less CPU resources than expected, then Oracle Database Resource Manager is not as effective at enforcing the resource allocations in the resource plan.
- Oracle Database is not supported for use with any UNIX operating system resource manager's memory management and allocation facility. Oracle Database Resource Manager, which provides resource allocation capabilities within an Oracle database instance, cannot be used with any operating system resource manager.

  For a complete list of operating system resource management and resource allocation and deallocation features that work with Oracle Database and Oracle Database Resource Manager, see your systems vendor and your Oracle representative. Oracle does not certify these system features for compatibility with specific release levels.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about Oracle Database Resource Manager
> - *Oracle Database Administrator's Guide* for information about instance caging

Resolving Operating System Issues

This section provides hints for tuning various systems by explaining the following topics:

- Performance Hints on UNIX-Based Systems
- Performance Hints on Windows Systems
- Performance Hints on HP OpenVMS Systems

Familiarize yourself with platform-specific issues so that you know what performance options the operating system provides.

> **See Also:**
>
> Your Oracle platform-specific documentation and your operating system vendor's documentation

Performance Hints on UNIX-Based Systems

On UNIX systems, try to establish a good ratio between the amount of time the operating system spends fulfilling system calls and doing process scheduling and the amount of time the application runs. The goal should be to run most of the time in application mode, also called user mode, rather than system mode.

The ratio of time spent in each mode is only a symptom of the underlying problem, which might involve the following:

- Paging or swapping
- Executing too many operating system calls
- Running too many processes

If such conditions exist, then there is less time available for the application to run. The more time you can release from the operating system side, the more transactions an application can perform.

Performance Hints on Windows Systems

On Windows systems, as with UNIX-based systems, establish an appropriate ratio between time in application mode and time in system mode. You can easily monitor many factors with the Windows administrative performance tool: CPU, network, I/O,

and memory are all displayed on the same graph to assist you in avoiding bottlenecks in any of these areas.

Performance Hints on HP OpenVMS Systems

Consider the paging parameters on a mainframe, and remember that Oracle Database can exploit a very large working set.

Free memory in HP OpenVMS environments is actually memory that is not mapped to any operating system process. On a busy system, free memory likely contains a page belonging to one or more currently active process. When that access occurs, a `soft page fault` takes place, and the page is included in the working set for the process. If the process cannot expand its working set, then one of the pages currently mapped by the process must be moved to the free set.

Any number of processes might have pages of shared memory within their working sets. The sum of the sizes of the working sets can thus markedly exceed the available memory. When the Oracle server is running, the SGA, the Oracle Database kernel code, and the Oracle Forms run-time executable are normally all sharable and account for perhaps 80% or 90% of the pages accessed.

Understanding CPU

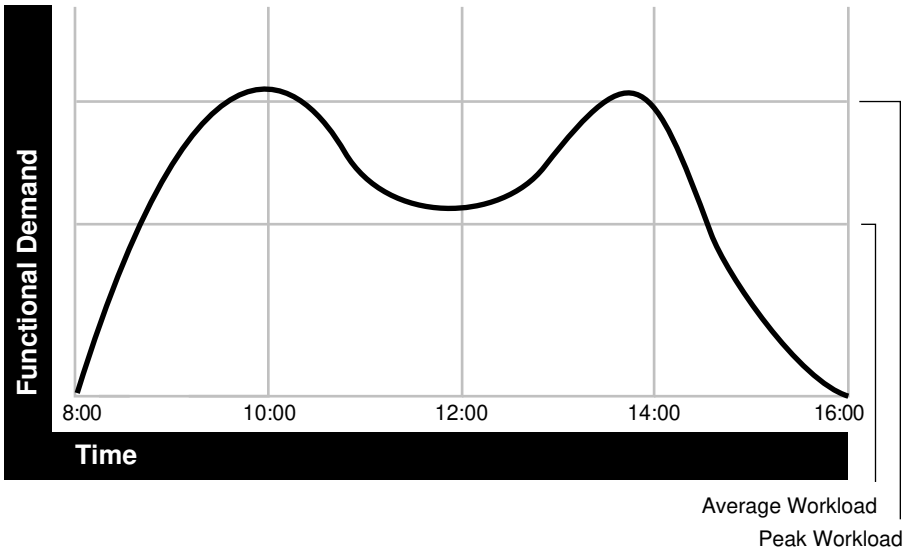
To address CPU problems, first establish appropriate expectations for the amount of CPU resources your system should be using. Then, determine whether sufficient CPU resources are available and recognize when your system is consuming too many resources. Begin by determining the amount of CPU resources the Oracle database instance utilizes with your system in the following three cases:

- System is idle, when little Oracle Database and non-Oracle activity exists
- System at average workloads
- System at peak workloads

You can capture various workload snapshots using the Automatic Workload Repository, Statspack, or the `UTLBSTAT`/`UTLESTAT` utility. Operating system utilities—such as `vmstat`, `sar`, and `iostat` on UNIX and the administrative performance monitoring tool on Windows—can be used along with the `V$OSSTAT` or `V$SYSMETRIC_HISTORY` view during the same time interval as Automatic Workload Repository, Statspack, or `UTLBSTAT`/`UTLESTAT` to provide a complimentary view of the overall statistics.

Workload is an important factor when evaluating your system's level of CPU utilization. During peak workload hours, 90% CPU utilization with 10% idle and waiting time can be acceptable. Even 30% utilization at a time of low workload can be understandable. However, if your system shows high utilization at normal workload, then there is no room for a peak workload. For example, The following figure illustrates workload over time for an application having peak periods at 10:00 AM and 2:00 PM.

Figure 18-1 Average Workload and Peak Workload



This example application has 100 users working 8 hours a day. Each user entering one transaction every 5 minutes translates into 9,600 transactions daily. Over an 8-hour period, the system must support 1,200 transactions an hour, which is an average of 20 transactions a minute. If the demand rate were constant, then you could build a system to meet this average workload.

However, usage patterns are not constant and in this context, 20 transactions a minute can be understood as merely a minimum requirement. If the peak rate you need to achieve is 120 transactions a minute, then you must configure a system that can support this peak workload.

For this example, assume that at peak workload, Oracle Database uses 90% of the CPU resource. For a period of average workload, then, Oracle Database uses no more than about 15% of the available CPU resource, as illustrated in the following equation:

```
20 tpm / 120 tpm * 90% = 15% of available CPU resource
```

where `tpm` is transactions a minute.

If the system requires 50% of the CPU resource to achieve 20 tpm, then a problem exists: the system cannot achieve 120 transactions a minute using 90% of the CPU. However, if you tuned this system so that it achieves 20 tpm using only 15% of the CPU, then, assuming linear scalability, the system might achieve 120 transactions a minute using 90% of the CPU resources.

As users are added to an application, the workload can rise to what had previously been peak levels. No further CPU capacity is then available for the new peak rate, which is actually higher than the previous.

Resolving CPU Issues

You can resolve CPU capacity issues by:

- Detecting and solving CPU problems from excessive consumption, as described in "Finding and Tuning CPU Utilization".
- Reducing the impact of peak load use patterns by prioritizing CPU resource allocation using Oracle Database Resource Manager, as described in "Managing CPU Resources Using Oracle Database Resource Manager".
- Using instance caging to limit the number of CPUs that a database instance can use simultaneously when running multiple database instances on a multi-CPU system, as described in "Managing CPU Resources Using Instance Caging".
- Increasing hardware capacity and improving the system architecture.

Finding and Tuning CPU Utilization

Every process running on your system affects the available CPU resources. Therefore, tuning non-database factors can also improve database performance.

Use the `V$OSSTAT` or `V$SYSMETRIC_HISTORY` view to monitor system utilization statistics from the operating system. Useful statistics contained in `V$OSSTAT` and `V$SYSMETRIC_HISTORY` include:

- Number of CPUs
- CPU utilization
- Load
- Paging
- Physical memory

> **See Also:**
>
> *Oracle Database Reference* for more information on `V$OSSTAT` and `V$SYSMETRIC_HISTORY`

You can use operating system monitoring tools to determine which processes run on the system as a whole. If the system is too heavily loaded, check the memory, I/O, and process management areas described later in this section.

You can use tools such as `sar -u` on many UNIX-based systems to examine the level of CPU utilization on the system. In UNIX, statistics show user time, system time, idle time, and time waiting for I/O. A CPU problem exists if idle time and time waiting for I/O are both close to zero (less than 5%) at a normal or low workload.

On Windows, you can use the administrative performance tool to monitor CPU utilization. This utility provides statistics on processor time, user time, privileged time, interrupt time, and DPC time.

Related topics:

- Checking Memory Management
- Checking I/O Management
- Checking Network Management
- Checking Process Management

> **Note:**
>
> This document describes how to check system CPU utilization on most UNIX-based and Windows systems. For other platforms, see your operating system documentation.

Checking Memory Management

Check the following memory management areas:

- Paging and Swapping
- Oversize Page Tables

Paging and Swapping

Use the `V$OSSTAT` view, utilities such as `sar` or `vmstat` on UNIX, or the administrative performance tool on Windows, to investigate the cause of paging and swapping.

Oversize Page Tables

On UNIX, if the processing space becomes too large, then it can result in the page tables becoming too large. This is not an issue on Windows systems.

Checking I/O Management

Thrashing is an I/O management issue. Ensure that your workload fits into memory, so the computer is not thrashing (swapping and paging processes in and out of memory). The operating system allocates fixed portions of time during which CPU resources are available to your process. If the process wastes a large portion of each time period checking to ensure that it can run and ensuring that all necessary components are in the computer, then the process might be using only 50% of the time allotted to actually perform work.

Checking Network Management

Check client/server round trips. There is an overhead in processing messages. When an application generates many messages that need to be sent through the network, the latency of sending a message can result in CPU overload. To alleviate this problem, bundle multiple messages rather than perform lots of round trips. For example, you can use array inserts, array fetches, and so on.

Checking Process Management

Several process management issues discussed in this section should be checked.

- Scheduling and Switching
- Context Switching
- Starting New Operating System Processes

Scheduling and Switching

The operating system can spend excessive time scheduling and switching processes. Examine the way in which you are using the operating system, because it is possible that too many processes are in use. On Windows systems, do not overload the server with too many non-database processes.

Context Switching

Due to operating system specific characteristics, your system could be spending a lot of time in context switches. Context switching can be expensive, especially with a large SGA. Context switching is not an issue on Windows, which has only one process for each instance. All threads share the same page table.

Oracle Database has several features for context switching:

- Post-wait driver

  An Oracle process must be able to post another Oracle process (give it a message) and also must be able to wait to be posted. For example, a foreground process may need to post LGWR to tell it to write out all blocks up to a given point so that it can acknowledge a commit.

  Often this post-wait mechanism is implemented through UNIX Semaphores, but these can be resource intensive. Therefore, some platforms supply a post-wait driver, typically a kernel device driver that is a lightweight method of implementing a post-wait interface.

- Memory-mapped system timer

  Oracle Database often needs to query the system time for timing information. This can involve an operating system call that incurs a relatively costly context switch. Some platforms implement a memory-mapped timer that uses an address within the processes virtual address space to contain the current time information. Reading the time from this memory-mapped timer is less expensive than the overhead of a context switch for a system call.

- List I/O interfaces to submit multiple asynchronous I/Os in One Call

  List I/O is an application programming interface that allows several asynchronous I/O requests to be submitted in a single system call, rather than submitting several I/O requests through separate system calls. The main benefit of this feature is to reduce the number of context switches required.

Starting New Operating System Processes

There is a high cost in starting new operating system processes. Developers often create a single-purpose process, exit the process, and then create a new one. This technique re-creates and destroys the process each time, consuming excessive amounts of CPU, especially in applications that have large SGAs. The CPU is needed to build the page tables each time. The problem is aggravated when you pin or lock shared memory because you must access every page.

For example, if you have a 1 gigabyte SGA, then you might have page table entries for every 4 KB, and a page table entry might be 8 bytes. You could end up with (1 GB / 4 KB) * 8 byte entries. This becomes expensive, because you need to continually ensure that the page table is loaded.

Managing CPU Resources Using Oracle Database Resource Manager

Oracle Database Resource Manager allocates and manages CPU resources among database users and applications in the following ways:

- Preventing CPU saturation

  If the CPUs run at 100%, then you can use Oracle Database Resource Manager to allocate a maximum amount of CPU to sessions in each consumer group. This feature can ensure that high-priority sessions can run immediately and lower the CPU consumption of low-priority sessions.

- Limiting CPU usage for a consumer group

  You can use the Resource Manager directive `max_utilization_limit` to place a hard limit on the percentage of CPU that a consumer group can use. This feature restricts the CPU consumption of low-priority sessions and can help provide more consistent performance for the workload in a consumer group.

- Limiting damage from runaway queries

  Starting with Oracle Database 11*g* Release 2 (11.2.0.2), Oracle Database Resource Manager can limit the damage from runaway queries by limiting the maximum execution time for a call, or by moving a long-running query to a lower-priority consumer group.

- Limiting the parallel statement activity for a consumer group

  Starting with Oracle Database 11*g* Release 2 (11.2.0.2), you can use the Resource Manager directive `parallel_target_percentage` to prevent one consumer group from monopolizing all parallel servers. The database queues parallel statements if they would cause this limit to be exceeded.

  For example, assume that the target number of parallel servers is 64, and the consumer group `ETL` has this directive set to 50%. If consumer group `ETL` is using 30 parallel servers, and if a new parallel statement needs 4 parallel servers, then the database would queue this statement.

See Also:

- *Oracle Database Administrator's Guide* to learn how to use Oracle Database Resource Manager
- *Oracle Database VLDB and Partitioning Guide* to learn how to use parallel query

Managing CPU Resources Using Instance Caging

When running multiple database instances on a single system, the instances compete for CPU resources. One resource-intensive database instance may significantly degrade the performance of the other instances. To avoid this problem, you can use instance caging to limit the number of CPUs that can used by each instance. Oracle Database Resource Manager then allocates CPU among the various database sessions according to the resource plan that you set for the instance, thereby minimizing the likelihood of the instance becoming CPU-bound.

See Also:

Oracle Database Administrator's Guide for information about using instance caging

Glossary

ADO policy

A policy that specifies a rule and condition for Automatic Data Optimization (ADO). For example, an ADO policy may specify that an object is marked `NOINMEMORY` (action) 30 days after creation (condition). Specify ADO policies using the `ILM` clause of `CREATE TABLE` and `ALTER TABLE` statements.

Automatic Data Optimization (ADO)

A technology that creates policies, and automates actions based on those policies, to implement an Information Lifecycle Management (ILM) strategy.

Automatic In-Memory

A feature that automatically evicts cold (infrequently accessed) segments from the IM column store to ensure that the working data set is always populated.

availability

The degree to which an application, service, or function is accessible on demand.

Bloom filter

A low-memory data structure that tests membership in a set. The database uses Bloom filters to improve the performance of hash joins.

Column Compression Unit (CU)

Contiguous storage for a column in an In-Memory Compression Unit (IMCU).

columnar data pool

The subpool in the In-Memory Area that stores columnar data. It is also known as the *1 MB pool*.

columnar format

The column-based format for objects that reside in the In-Memory Column Store. The columnar format contrasts with the row format used in data blocks.

common dictionary

A segment-level, instance-specific set of master dictionary codes, created from local dictionaries. A local dictionary is a sorted list of dictionary codes specific to a Column Compression Unit (CU). A join group uses a common dictionary to optimize joins.

compression tiering

The application of different levels of compression to data based on its access pattern. For example, administrators may compress inactive data at a higher rate of compression at the cost of slower access.

data flow operator (DFO)

The unit of work between data redistribution stages in a parallel query.

database buffer cache

The portion of the #unique_616 that holds copies of data blocks. All client processes concurrently connected to the #unique_617 share access to the buffer cache.

dense grouping key

A key that represents all grouping keys whose grouping columns come from a specific fact table or dimension.

dense join key

A key that represents all join keys whose join columns come from a particular fact table or dimension.

dense key

A numeric key that is stored as a native integer and has a range of values.

double buffering

A repopulation mechanism in which background processes create new In-Memory Compression Unit (IMCU) versions by combining the original rows with the latest modified rows. During repopulation, the stale IMCUs remain accessible for queries.

expression

A combination of one or more values, operators, and SQL functions that resolves to a value.

expression capture interval

The time interval within which the database considers IM expressions for possible capture.

expression capture window

An expression capture interval defined by invocation of the `IME_OPEN_CAPTURE_WINDOW` and `IME_OPEN_CAPTURE_WINDOW` procedures in the `DBMS_INMEMORY_ADMIN` package.

Expression Statistics Store (ESS)

A repository maintained by the optimizer to store statistics about expression evaluation. For each segment, the ESS monitors statistics such as frequency of execution, cost of evaluation, timestamp evaluation, and so on. The ESS is persistent in nature and has an SGA representation for fast lookup of expressions.

Heat Map

Heat Map shows the popularity of data blocks and rows. Automatic Data Optimization (ADO) to decide which segments are candidates for movement to a different storage tier.

home location

The database instance in which an IMCU resides. When auto DOP is enabled on Oracle RAC, the parallel query coordinator uses home location to determine where each IMCU is located, how large it is, and so on.

In-Memory hybrid scan

A query that scans both the IM column store and the row store. The optimizer considers an In-Memory hybrid scan automatically when all predicate columns have the `INMEMORY` attribute, and some columns in the `SELECT` list do not have the `INMEMORY` attribute.

hybrid partitioned table

A table in which some partitions are stored in data file segments and some are stored in external data source.

IM aggregation

An optimization that accelerates aggregation for queries that join from a single large table to multiple small tables. The transformation uses `KEY VECTOR` and `VECTOR GROUP BY` operators, which is why it is also known as *`VECTOR GROUP BY` aggregation*.

IM column store

An optional SGA area that stores copies of tables and partitions in a columnar format optimized for rapid scans.

IM dynamic scan

The use of lightweight threads to automatically parallelize In-Memory table scans.

IM expression

A SQL expression whose results are stored in the In-Memory Column Store. If `last_name` is a column stored in the IM column store, then an IM expression might be `UPPER(last_name)`.

IMCU mirroring

In Oracle RAC, the duplication of an IMCU in multiple IM column stores. For example, the IM column stores on instance 1 and instance 2 are populated with the same `sales` table.

IMCU pruning

In a query of the In-Memory Column Store, the elimination of IMCUs based on the high and low values in each IMCU. For example, if a statements filters product IDs greater than 100, then the database avoids scanning IMCUs that contain values less than 100.

IM storage index

A data structure in an IMCU header that stores the minimum and maximum for all columns within the IMCU.

In-Memory Advisor

A downloadable PL/SQL package that analyzes the analytical processing workload in your database. This advisor recommends a size for the IM column store and a list of objects that would benefit from In-Memory population.

In-Memory Aggregation

See IM aggregation.

In-Memory Area

An optional SGA component that contains the IM column store.

In-Memory Column Store

See IM column store.

In-Memory Compression Unit (IMCU)

A storage unit in the In-Memory Column Store that is optimized for faster scans. The In-Memory Column Store stores each column in table separately and compresses it. Each IMCU contains all columns for a subset of rows in a specific table segment.
A one-to-many mapping exists between an IMCU and a set of database blocks. For example, if a table contains columns `c1` and `c2`, and if its rows are stored in 100 database blocks on disk, then IMCU 1 might store the values for both columns for blocks 1-50, and IMCU 2 might store the values for both columns for blocks 51-100.

In-Memory Coordinator Process (IMCO)

A background process whose primary task is to initiate background population and repopulation of columnar data.

In-Memory Dynamic Scan

See IM dynamic scan.

In-Memory Expression

See IM expression.

In-Memory Expression Unit (IMEU)

A container that stores the computed result of an In-Memory Expression (IM expression). Each IMEU is linked to its own parent In-Memory Compression Unit (IMCU).

In-Memory FastStart

A feature that significantly reduces the time to populate data into the IM column store when a database instance restarts.

In-Memory population

See population.

In-Memory virtual column

A virtual column that is eligible to be populated in the In-Memory Column Store.

Information Lifecycle Management (ILM)

A set of processes and policies for managing data throughout its useful life.

join group

A user-defined object that specifies frequently joined columns from the same table or different tables. External tables are not supported.

A typical join group candidate is a set of columns used to join fact and dimension tables. Join groups are only supported when `INMEMORY_SIZE` is a nonzero value.

key vector

A data structure that maps between dense join keys and dense grouping keys.

large pool

Optional area in the SGA that provides large memory allocations for backup and restore operations, I/O server processes, and session memory for the shared server and Oracle XA.

local dictionary

A sorted list of dictionary codes specific to a Column Compression Unit (CU).

lightweight thread

An execution entity used in an In-Memory Dynamic Scan. Lightweight threads help to parallelize scans of IMCUs.

metadata pool

A subpool of the In-Memory Area that stores metadata about the objects that reside in the IM column store. The metadata pool is also known as the *64 KB pool*.

memoptimize pool

An SGA pool that stores buffers and related structures for heap-organized tables specified as `MEMOPTIMIZE FOR READ`.

on-demand population

When `INMEMORY PRIORITY` is set to `NONE`, the IM column store *only* populates the object when it is accessed through a full scan. If the object is never accessed, or if it is accessed only through an index scan or fetch by rowid, then it is never populated.

OSON

Oracle's optimized binary JSON format. OSON enables fast queries and updates of the JSON data model in Oracle database server and Oracle database clients.

OZIP

A proprietary compression technique that offers extremely fast decompression. OZIP is tuned specifically for Oracle Database.

partition exchange load

A technique in which you create a table, load data into it, and then exchange an existing table partition with the table. This exchange process is a DDL operation with no actual data movement.

population

The operation of reading existing data blocks from data files, transforming the rows into columnar format, and then writing the columnar data to the IM column store. In contrast, *loading* refers to bringing new data into the database using DML or DDL.

priority-based population

When `PRIORITY` is set to a value other than `NONE`, Oracle Database adds the object to a prioritized population queue. The database populates objects based on their queue position, from `CRITICAL` to `LOW`. It is "priority-based" because the IM column store automatically populates objects using the prioritized list whenever the database re-opens. Unlike in on-demand population, objects do not require a full scan to be populated.

repopulation

The automatic refresh of a currently populated In-Memory Compression Unit (IMCU) after its data has been significantly modified. In contrast, population is the initial creation of IMCUs in the IM column store.

service

The logical representation of an application workload that shares common attributes, performance thresholds, and priorities. A single service can be associated with one or more instances of an Oracle RAC database, and a single instance can support multiple services.

SIMD

Single Instruction, Multiple Data. An instruction that processes data as a single unit, called a **vector**, rather than as separate instructions. SIMD processing is known as **vectorization**.

Snapshot Metadata Unit (SMU)

A storage unit in the In-Memory Area that contains metadata and transactional information for an associated In-Memory Compression Unit (IMCU).

Space Management Worker Process (W*nnn*)

A process that populates or repopulates data in the IM column store on behalf of In-Memory Coordinator Process (IMCO).

staleness threshold

An internally set percentage of entries in the transaction journal for an IMCU that initiates repopulation.

storage tiering

The deployment of data on different tiers of storage depending on its level of access. For example, administrators migrate inactive data from high-performance, high-cost storage to low-cost storage.

table scan process

A foreground or PQ process that coordinates an IM dynamic scan.

threshold-based repopulation

The automatic repopulation of an IMCU when the number of stale entries in an IMCU reaches an internal staleness threshold.

transaction journal

Metadata in a Snapshot Metadata Unit (SMU) that keeps the IM column store transactionally consistent.

trickle repopulation

A supplement to threshold-based repopulation. The In-Memory Coordinator Process (IMCO) may instigate trickle repopulation automatically for any IMCU in the IM column store that has stale entries but does not meet the staleness threshold.

vector aggregation

See IM aggregation.

virtual column

A column that is not stored on disk. The database derives the values in virtual columns on demand by computing a set of expressions or functions.

working data set

The subset of `INMEMORY` objects that is actively queried at a given time. Typically, the work working data set changes over time.

Index

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

W