Database Performance with Oracle Database 10g Release 2

An Oracle White Paper May 2005

Introduction	5
Manageability of performance	5
Self-tuning Database	5
Automatic Workload Repository (AWR)	5
Automatic Database Diagnostic Monitor (ADDM)	
Automatic SQL Tuning	6
SQL Access Advisor	7
Self-tuning SGA	7
Self-tuning checkpoints	7
Self-tuning Multi-block read count	
Managing large populations of concurrent users	
Removing internal contention for shared resources	9
Multi-Version Read Consistency	
Non-Escalating Row-Level Locking	
Impact of locking on packaged applications	.10
Asynchronous commit	.11
Support for complex transactions	.11
Rich Query Processing Techniques	.11
Indexes	.11
Index-Organized Tables	13
Cost-Based Query Optimization	13
Sorted Hash clusters	
Sophisticated Summary Management	.17
Optimized PL/SQL	.19
Managing large volumes of data	.20
Support for Parallel Execution	.20
Parallel single cursor	21
Data Partitioning	
Rolling window operations	.22
Partition Pruning	.23
Partition-Wise Joins	.24
Global indexes	.25
Large number of partitions	.26
Full Table Scans	.27
In-memory Sorts	.27
Hash-based Aggregation	.27
High-speed Data Movements	
Summary of Oracle Database 10g's new features for performance	.29

Database Performance with Oracle Database 10g

INTRODUCTION

Oracle Database supports many of the largest information systems in the world. For nearly 3 decades it has set the industry standard for performance and scalability. Starting with multi-version read consistency in Oracle6, each release has introduced innovative features designed to improve database performance and scalability. And each release has run essentially unchanged on all major platforms available at the time. This paper covers those aspects of Oracle Database that relate directly to performance and scalability, with a focus on the new features and performance improvements in the latest release, Oracle Database 10g Release 2.

Database performance challenges fall into three categories: accommodating extremely large populations of concurrent users, handling increasingly complex transactions, and supporting ultra large databases. All these challenges have to be met while ensuring that performance diagnosis and database tuning do not consume significant amounts of time and effort. Excellent performance must be easy to deliver and maintain, at the lowest cost possible.

In this paper we see how Oracle Database handles these challenges.

MANAGEABILITY OF PERFORMANCE

Self-tuning Database

Oracle Database 10g provides several features that simplify performance monitoring and automate the detection and resolution of performance problems. With these features, Oracle Database becomes the first truly self-tuning database¹.

Automatic Workload Repository (AWR)

The Automatic Workload Repository (AWR) is a persistent repository within the Oracle Database that contains performance data and statistics about the operations

¹ For more information about Oracle Database 10g's self-manageability features, see "Oracle Database 10g: The Self-Managing Database". an Oracle white paper, November 2003

of the database. At regular intervals, Oracle Database takes a snapshot of all its vital statistics and workload information and stores them in the AWR. These statistics provide the data for the diagnostic facilities of Oracle Database 10g that support both pro-active and reactive monitoring².

Automatic Database Diagnostic Monitor (ADDM)

ADDM is a self-diagnosis engine in the database that proactively analyzes data captured in AWR to understand the state of the system. The goal of ADDM is to identify those parts of the system that are consuming the most 'DB time', and to reduce this time whenever possible, either by recommending solutions, or by referring to other advisory components, such as the new SQL Access Advisor. ADDM drills down to identify the root cause of problems rather than focusing just on the symptoms and reports the overall impact of the problem. ADDM also quantifies the expected benefits from its recommendations and documents areas of the system that do not exhibit slow performance and so do not need tuning.

Automatic SQL Tuning

Oracle Database 10g completely automates the tuning of SQL statements³.

Automatic SQL Tuning is based on the Automatic Tuning Optimizer. In automatic tuning mode, the query optimizer is given more time to perform the investigation and verification steps required for the tuning process. This additional time allows the optimizer to use techniques, such as dynamic sampling or partial execution, which could not be used under the time constraints of the regular operating mode. These techniques help the optimizer validate its own estimates of cost, selectivity and cardinality. As a result, using the automatic tuning mode increases the probability of generating well-tuned plans for SQL statements.

The functionality of the Automatic Tuning Optimizer is exposed via the SQL Tuning Advisor. The Automatic Tuning Optimizer performs multiple analyses: statistics analysis, SQL profiling, access path analysis, and SQL structure analysis⁴.

The conclusions generated by the Automatic Tuning Optimizer are conveyed to users via the SQL Tuning Advisor in the form of a tuning advice. The advice consists of one or more recommendations, each with a rationale and an estimated benefit obtained when implemented. Such advice could include a recommendation for adding new indexes, rewriting the SQL statement, or implementing SQL profiles. Users are given the option to accept the advice, thus completing the tuning of the corresponding SQL statement.

Performance with Oracle Database 10G

Page

 $^{^2}$ For more information about AWR and ADDM, see "The Self-Managing Database: Automatic Performance Diagnosis", an Oracle white paper, November 2003

³ For more information about Automatic SQL Tuning, see "The Self-Managing Database: Guided Application & SQL Tuning", an Oracle white paper, November 2003.

⁴ These analyses and their results are described in detail in the Oracle white paper "The Self-Managing Database: Guided Application & SQL Tuning", November 2003

SQL Access Advisor

The SQL Access Advisor automatically analyzes the schema design for a given workload and recommends indexes and materialized views to create, retain or drop as appropriate for the workload. While generating recommendations, the SQL Access Advisor considers the impact of adding new indexes and materialized views on data manipulation activities, such as insert, update and delete, in addition to the performance improvement they are likely to provide for queries.

Self-tuning SGA

The System Global Area (SGA) is the memory region, shared by all users, that contains data and control information for a particular Oracle Database instance. The SGA is internally divided into memory components that represent pools of memory used to satisfy each category of memory allocation requests, such as requests for storing the most recently used blocks of data or logging the changes made to the database.

Tuning the sizes of these caches for optimal performance is not an easy task, even with advisory mechanisms. There is always the risk of either under-sizing a component, leading to memory allocation failures, or over-sizing a component, which wastes memory that could be used by other caches.

The first release of Oracle Database 10g introduced the self-tuning SGA feature that allows administrators to specify only the total size of the SGA (SGA_TARGET), and leaves to Oracle Database the responsibility to internally determine the optimal distribution of memory across the various SGA pools. With this new feature, distribution of memory to the various SGA caches changes dynamically over time to accommodate changes in the workload. Oracle Database 10g Release 2 further eases the task of administrators by providing the new SGA Size Advisor that can be used to get advice on optimal settings of the SGA_TARGET parameter.

Together with the parameter PGA_AGGREGATE_TARGET, introduced with Oracle9*i* Database, self-tuning SGA enables Oracle Database to automatically and dynamically adjust memory consumption to changes in workload distributions, guaranteeing optimal memory usage.

Self-tuning checkpoints

Checkpoints are means of synchronizing the data modified in memory with the data files of the database. By periodically writing modified data to the data files between checkpoints Oracle Database ensures that sufficient amounts of memory are available, improving the performance of finding free memory for incoming operations.

 $^{^{\}rm 5}$ For more information about Oracle and Grid computing, see Oracle and the Grid, An Oracle white paper, November 2002

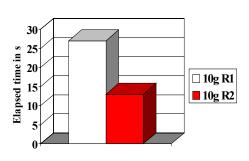


Figure 1: Object Checkpointing

Prior to Oracle Database 10g administrators could specify the expected crash recovery time (MTTR) by setting the value of a checkpoint-related initialization parameter (FAST_START_MTTR_TARGET). They could do so by using the MTTR advisory, which helps predict the number of physical writes that would arise with different MTTR target values. With Oracle Database 10g, the database can self-tune checkpoints activity to achieve good recovery times with low impact on normal throughput. With automatic checkpoint tuning, Oracle Database takes advantage of periods of low I/O usage to write out data modified in memory to the data files without adverse impact on the throughput. Consequently, a reasonable crash recovery time can be achieved even if the administrator does not set any checkpoint-related parameter or if this parameter is set to a very large value.

Another enhancement done in the second release of Oracle Database 10g dramatically improves the performance⁶ of object-checkpoint requests issued for objects accessed through direct path reads, a situation that can occur with parallel query. Before an object can be accessed through direct path reads, dirty buffers of the object must be written to data files on disk via an object-checkpoint request. Prior to Oracle Database 10g Release 2, the checkpoint request is handled by issuing a checkpoint for the tablespace the object belongs to, writing out all the dirty buffers for the entire tablespace. Since a large number of objects may reside in the same tablespace, this implementation may cause large number of unnecessary disk writes. With the new release, a checkpoint request for a target object will only write out the dirty buffers of that object, without incurring any additional writes for the dirty buffers of other objects.

Self-tuning Multi-block read count

Scan operations, such as full table scans or index fast full scans, read all the blocks of the object. The cost of reading the blocks from disk into the buffer cache can be amortized by reading the blocks in large I/O operations. The DB_FILE_MULTIBLOCK_READ_COUNT parameter controls the number of blocks that are pre-fetched into the buffer cache if a cache miss is encountered for a particular block. The value of this parameter can have a significant impact on the overall database performance and it is not easy for the administrator to determine its most appropriate value.

Oracle Database 10g Release 2 automatically selects the appropriate value for this parameter depending on the operating system optimal I/O size and the size of the buffer cache.

MANAGING LARGE POPULATIONS OF CONCURRENT USERS

Database servers face the challenge of servicing an unpredictable (potentially extremely large) number of users, while at the same time providing predictable and satisfying performance for all types of queries and updates, both batch and real-

⁶ For further details on the tests results for Oracle Database 10g Release 2, see DSS performance in Oracle Database 10g Release 2, an Oracle white paper, May 2005.

time. Oracle Database 10g's built-in capabilities address these requirements, guaranteeing excellent scalability and optimal resource utilization at all times.

Removing internal contention for shared resources

The basic architecture of Oracle Database is very efficient for managing large numbers of concurrent transactions. The technical features that make this possible, multi-version read consistency and patented non-escalating row-level locking, ensure that as more users are connected to an application, Oracle Database can continue to deliver consistent performance despite the increasing volume of transactions.

Multi-Version Read Consistency

Database implementations differ in their ability to prevent well-known phenomena encountered in multi-user environments:

- dirty, or uncommitted, reads happen when a transaction can read changes made to the database that have not yet been committed.
- non-repeatable reads occur when a transaction re-reads data it has previously read and finds that another committed transaction has modified or deleted the data.
- phantom reads happen when a transaction executes a query twice returning a
 set of rows satisfying a search condition, and finds that the second query can
 retrieve additional rows which were not returned by the first query, because
 other applications were able to insert rows that satisfy the condition.

Oracle Database's implementation of multi-version read consistency always provides consistent and accurate results. When a transaction updates data, the original data values are recorded in the database's undo records. Oracle Database uses these values to construct a read-consistent view of a table's data, and to ensure that a version of the information, consistent at the beginning of the uncommitted transaction, can always be returned to any user.

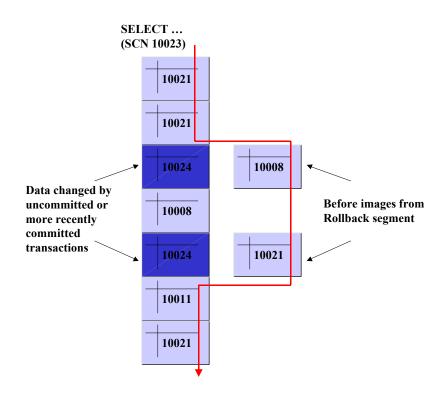


Figure 2: Transactions and multi-version read consistency. Red line represents data values shown to transaction 10023

As a result, Oracle Database fully supports mixed workloads environments characterized by simultaneous query and update activities. With Oracle Database, writers never block readers and readers never block writers. Non-blocking multiversion read consistency always provides users with consistent query results while never imposing a performance penalty on concurrent update activity.

Non-Escalating Row-Level Locking

Row-level locks offer the finest granularity of lock management, and, therefore, the highest degree of data concurrency. Row-level locking ensures that any user or operation updating a row in a table will only lock that row, leaving all other rows available for concurrent operations.

Oracle Database uses row-level locking as the default concurrency model. Because it stores locking information with row data, Oracle Database can have as many row level locks as there are rows or index entries in the database, thus providing unlimited data concurrency. As a consequence, Oracle Database never has to escalate locks and Oracle users never experience deadlock situations due to lock escalation.

Impact of locking on packaged applications

The locking schemes in a database are transparently applied for all applications that use the database. Workarounds for weaknesses in these locking schemes have to

be implemented with additional application code. This issue does not exist with Oracle Database whose powerful multi-version read consistency model provides scalability benefits to all applications. In contrast, SAP had to implement special features in the application mid-tier to handle the limited number of locks available and the possibility of deadlock in DB2. Oracle Database's concurrency control architecture is the major reason why most enterprise packaged applications, including SAP R/3, PeopleSoft and Siebel, have the majority of their installed base running on Oracle Database.

Asynchronous commit

When a transaction is committed, Oracle Database guarantees that the commit is durable, meaning that the commit information is flushed to disk in the online redo log files before the call is returned to the client, ensuring that the transaction persists across database crashes. This introduces a slight latency in the commit operation, since the call is only returned when the commit is persisted on disk.

Asynchronous commit gives application developers more control over the behavior of the transaction commit, giving them the options to force or not force the redo log I/O and to wait or not wait for the commit to be persisted on disk. This reduces commit latency and offers better transaction throughput for applications that are willing to tolerate a small window of the non-durability of a commit.

SUPPORT FOR COMPLEX TRANSACTIONS

Rich Query Processing Techniques

Oracle Database offers a rich variety of query processing techniques that address the requirements of very complex environments. These sophisticated techniques include:

- Indexing techniques and schema objects tailored for all kinds of applications
- Cost-based query optimization for efficient data access
- Summary management capabilities

Indexes

Indexes are database structures created to provide a faster path to data. Using indexes can dramatically reduce disk I/O operations, thus increasing the performance of data retrieval. Oracle Database supports the industry's largest selection of indexing schemes:

- B-Tree indexes
- B-Tree cluster indexes
- Hash Cluster Indexes
- Reverse Key Indexes

- Bitmap Indexes
- Bitmap Join Indexes
- Function-based indexes
- Domain Indexes

The presence or absence of indexes is completely transparent to applications.

Oracle Database supports static bitmap indexes⁷ and bitmap join indexes⁸, whose usage can provide huge performance benefits for typical load and query operations in data warehousing environments. These indexing schemes are strong differentiators between Oracle Database and IBM DB2 or Microsoft SQL Server and are explained in more detail in the following section.

Bitmap Indexes & Bitmap Join Indexes

A bitmap index uses a bitmap (or bit vector) for each key value instead of a list of the table rows' storage locations. Each bit in the bitmap corresponds to a row in the table. The bit is set when the table's row contains the key value.

Bitmap representation can save a lot of space over lists of rows' storage locations, especially for low cardinality data. The table in figure 3 compares the respective sizes of bitmap indexes vs. traditional b-tree indexes when the cardinality of the index varies.

Bitmap indexes lend themselves to fast Boolean operations for combining bitmaps from different index entries. Bitmap indexing efficiently merges indexes that correspond to several conditions in a WHERE clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time, often dramatically⁹.

A bitmap join index is a bitmap index for the join of two or more tables. A bitmap join index can be used to avoid actual joins of tables, or to greatly reduce the volume of data that must be joined, by performing restrictions in advance. Queries using bitmap join indexes can be sped up via bit-wise operations.

Bitmap join indexes, which contain multiple dimension tables, can eliminate bitwise operations, which are necessary in the star transformation with bitmap indexes on single tables. Performance measurements performed under various types of star queries demonstrate tremendous response time improvements when queries use bitmap join indexes.

⁷ since Oracle7.3

⁸ since Oracle9i

⁹ See Key Data Warehousing Features in Oracle9i: A Comparative Performance Analysis, An Oracle White Paper, September 2001 http://otn.oracle.com/deploy/performance/content.html

Index-Organized Tables

Index-organized tables¹⁰ provide fast access to table data for queries involving exact match and/or range search on the primary key because indexing information and data are stored together. Use of index-organized tables reduces storage requirements because the key columns are not duplicated in both the table and the primary key index. It eliminates the additional storage required for rows' storage locations that are used in conventional indexes to link the index values and the row data. As a result, performance is increased because the access time necessary to retrieve data is reduced.

Index-organized tables support full table functionality, including partitioning and parallel query.

Cost-Based Query Optimization

Query optimization is of great importance for the performance of a relational database, especially for the execution of complex SQL statements. Oracle Database uses a cost-based optimization strategy¹¹. With cost-based optimization, multiple execution plans are generated for a given query, and an estimated cost is computed for each plan. The query optimizer then chooses the best plan, that is, the plan with the lowest estimated cost. This query optimization process is entirely transparent to the application and the end-user.

Because applications may generate very complex SQL code, query optimizers must be extremely sophisticated and robust to ensure good performance. Oracle Database's cost-based optimizer produces excellent execution plans thanks to the accuracy and completeness of its cost model and the techniques and methods it uses to determine the most efficient means of accessing the specified data for a particular query¹².

SQL Transformation

The optimizer can transform the original SQL statement into a SQL statement that returns the same result, but can be processed more efficiently. Heuristic query transformations, such as view merging or predicate pushing, are applied whenever possible because they always improve performance of queries by greatly reducing the amount of data to be scanned, joined, or aggregated. Oracle Database also applies cost-based query transformation, where decisions to transform queries, using various techniques such as materialized view rewrite or star transformation, are based on the optimizer's cost estimates.

¹⁰ Already available with Oracle9i Database

¹¹ Starting with Oracle Database 10G the rule-based optimizer is no longer supported

¹² For more details about query optimization with Oracle, see Query Optimization in Oracle9i, An Oracle white paper, February 2002

Execution plan selection

The execution plan describes all the execution steps of the SQL statement processing, such as the order in which tables are accessed, how the tables are joined together and whether these tables are accessed via indexes or not.

Oracle Database provides an extremely rich selection of database structures, partitioning and indexing techniques, and join methods. Its parallel execution architecture allows virtually any SQL statement to be executed with any degree of parallelism. Additionally, the cost-based optimizer considers hints, which are optimization suggestions driven by users and placed as comments in the SQL statement.

Many different plans can be generated by the optimizer for a given SQL statement because of the variety of combinations of different access paths, join methods, and join orders that can be used to access and process data in different ways and produce the same result.

Cost estimate

To estimate the cost of these execution plans, and chose the plan with the lowest cost, the optimizer relies upon cost estimates for the individual operations that make up the execution of the SQL statement. These estimates have to be as accurate as possible and Oracle Database integrates a very sophisticated cost model that factors in-depth knowledge about Oracle Database's data structures and access methods with object-level and system statistics and performance information.

Object-level statistics gather information about the objects in the database (tables, indexes, and materialized views), such as the number of levels in a b-tree index or the number of distinct values in a table's column (cardinality). Data value histograms can also be used to get accurate estimates of the distribution of column data.

System statistics describe the performance characteristics of the hardware components (processors, memory, storage, network) collected during the activities of typical workloads.

In Oracle Database10g, the default cost model is "CPU+IO". In order to estimate the execution time of a given query, the query optimizer estimates the number and type of IO operations as well as the number of CPU cycles the database will perform during the execution of the query. It then uses system statistics to convert these numbers of CPU cycles and IO operations into execution time. This improved cost model results in better execution plans and, thus, improved performance for some types of queries. In those cases, the improvement factor compared to the "IO" model can be up to 77% (elapsed time).

In the first example illustrated in figure 3, a join query is performed on the Sales and Products tables of the Oracle Sample schema. The query checks that the referential integrity constraint between the sales and products tables is enforced. With the IO cost model, the query optimizer will only consider the cost of scanning

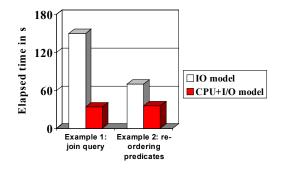


Figure 3: examples of performance improvements with the "CPU+IO" cost model

the Sales table and will choose a nested loop for the execution plan. With the CPU+IO cost model, the cost of the operations performed in memory is taken into account and, as a result, the optimizer chooses a hash join instead. The result is a much better execution plan, with better performance.

Another potential benefit of the "CPU+IO" cost model is the ability to re-order predicates in queries. Predicate re-ordering is made possible only when the cost of each predicate can be evaluated, which is only possible in the "CPU+IO" cost model because, in most cases, the cost of a predicate only contains CPU cycles. The query used for the second example has two predicates, with a much higher cost of execution for the first predicate. With IO cost model, the predicates are executed in the order of the original query. With CPU+IO cost model, the predicates are re-ordered so that the predicate with the lower cost is executed first. With this re-ordering, the execution of the second predicate is skipped for rows that do not satisfy the first condition, resulting in better performance¹³.

The process for gathering statistics and performance information needs to be both highly efficient and highly automated. Oracle Database 10g introduces automatic statistics collection. Objects with stale or no statistics are automatically analyzed, relieving the administrator from the task of keeping track of what does and what does not need to be analyzed, and then analyzing them as needed. Fully automated statistics collection subsequently improves SQL execution performance.

Oracle Database uses sampling to gather statistics by examining relevant samples of data. Sampling can be static or dynamic, occurring in the same transaction as the query, and can be used in conjunction with parallelism. Oracle Database's statistics gathering routines automatically determines the appropriate sampling percentage as well as the appropriate degree of parallelism, based upon the data-characteristics of the underlying table. Oracle Database also implicitly determines which columns require histograms, which are used to get accurate estimates of the distribution of column data.

Users can influence the optimizer's choices by setting the optimizer approach and goal. Oracle Database provides two optimizer modes. The first mode minimizes the time to return the first *n* rows of query. This mode corresponds to applications, such as operational systems, where the goal is to get the best response time to return the first rows. The second mode is used to minimize the time to return all of the rows from a query, with a goal of best throughput.

Dynamic runtime optimization

As not every aspect of SQL execution can be optimally planned ahead of time, Oracle Database makes run-time dynamic adjustments to its query-processing strategies based on business priorities and current database workload and hardware

http://otn.oracle.com/products/bi/db/10g/pdf/twp_dss_performance_10gr1_0903.pdf

Performance with Oracle Database 10G

Page

 $^{^{13}}$ For a detailed explanation of these examples, see "DSS Performance in Oracle Database 10g", an Oracle white paper, September 2003, at

capabilities. The goal of these dynamic optimizations is to achieve optimal performance even when each query may not be able to obtain the ideal amount of CPU or memory resources.

Oracle Database automatically adjusts the degree of parallelism of queries, dynamically allocates appropriate amounts of memory to each individual query, and uses the Resource Manager to allocate resources among queries based on the directives specified by resource plans.

The result is an improved accuracy of the cost and size models used by the query optimizer. This helps the optimizer produce better execution plans, improving query performance.

Sorted Hash clusters

Oracle offers several optional methods for storing table data, including clusters and hash clusters. Oracle Database 10g adds a new option called sorted hash clusters, which is particularly suitable for systems requiring very high data insertion and retrieval rates.

This table structure better supports first-in first-out (FIFO) type data processing applications, where data is processed in the order it was inserted. Such applications are typically found in the telecommunications and manufacturing environments. A good example is the call detail record (CDR) data structure for a telecommunications switch. There is a fixed space of telephone numbers for call origination, and a potentially unlimited number of calls from each telephone number that pass through the switch. For billing and auditing purposes, each call must be recorded and associated with its origin. Calls are stored as they fly through the switch, and retrieved later in FIFO order when customer bills are generated.

This is illustrated in the figure below.

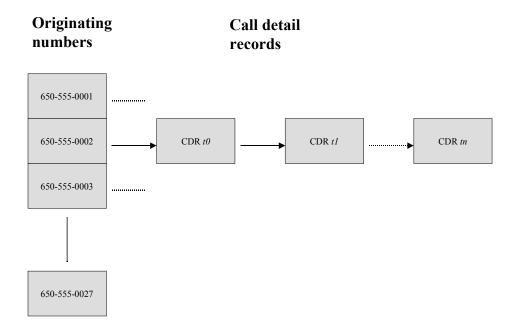


Figure 4: Call Detail Record data structure

While it is possible to capture this data in two standard tables, a sorted hash cluster is a better solution as it allows inexpensive access to the sorted list of call records for any given subscriber, and allows the billing application reading the call records to walk through the sorted list in a FIFO manner.

Any application in which data is always consumed in the order in which it is inserted will see tremendous performance improvements by using this new type of optimized table structure.

Sophisticated Summary Management

Materialized Views

Materialized views are schema objects that can be used to summarize, pre-compute, replicate, and distribute data. They are suitable in various computing environments such as data warehousing, decision support, and distributed or mobile computing. In a data warehousing application, for example, users often issue queries that summarize detail data by common dimensions, such as month, product, or region. Materialized views provide the mechanism for storing these multiple dimensions and summary calculations. The utilization of materialized views by the cost-based optimizer can result in dramatic improvements in query performance (see next section on query rewrite).

Materialized views must be refreshed when the data in their master tables changes. Complete refresh re-executes the materialized view query, thereby completely recomputing the contents of the materialized view from the master tables. Because complete refresh can be extremely long, many data warehouse environments require fast, incremental refresh in order to meet their operational objectives.

Fast refresh uses a variety of incremental algorithms to update the materialized view to take into account the new and updated data in the master tables. Oracle Database provides conventional fast refresh mechanisms, which are used when conventional DML operations, such as UPDATE, or direct load operations are executed against the master tables, and partition-aware fast-refresh mechanisms, which follow maintenance operations or DML changes on the partitions of the base tables. For instance, if a base table's partition is truncated or dropped, the affected rows in the materialized view are identified and deleted.

Oracle Database 10g broadens support for fast refresh mechanisms to a wider variety of materialized views. With this release, partition-aware fast refresh has been extended to materialized views whose base tables are list-partitioned or use ROWID as a partition marker.

Oracle Database 10g also extends fast refresh by utilizing functional dependencies and query rewrite. When users define materialized views along dimensional hierarchies, Oracle Database discovers the affected partitions in the materialized view corresponding to the affected partitions in the base tables and generates efficient refresh expressions going against other materialized views or base tables.

Figure 5 shows performance improvements obtained by using incremental refresh and query rewrite mechanisms to refresh materialized views following modifications on the partitions of the base tables¹⁴.



Query rewrite is the query optimization technique that transforms a user query written in terms of tables and views to execute faster by fetching data from materialized views.

When a query requests a summary of detail records, the query optimizer automatically recognizes when an existing materialized view can and should be used to satisfy the request. The optimizer transparently re-writes the query to use the materialized view instead of the underlying tables. This results in dramatic improvements in query performance because the materialized view has precomputed join and aggregation operations on the database prior to execution and stored the results in the database. Rewriting the query to use a materialized view avoids the summing of the detail records every time the query is issued.

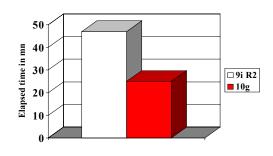


Figure 5: Partition-aware Materialized Views Fast-Refresh

¹⁴ For a detailed explanation of these examples, see "DSS Performance in Oracle Database 10g", an Oracle white paper, September 2003, at http://otn.oracle.com/products/bi/db/10g/pdf/twp dss performance 10gr1 0903.pdf

Oracle Database has a very robust set of rewrite techniques for materialized views, in order to allow each materialized view to be used for as broad a set of queries as possible.

With Oracle Database 10g, query rewrite can use more than one materialized view. As a result, more queries are now eligible for query rewrite and are likely to experience improved query response time.

Summary Advisor

Oracle Database's summary advisor frees administrators from time-consuming tuning and diagnostics tasks and allows them to simulate a variety of "what-if" scenarios.

The summary advisor is a collection of functions and procedures that provide analysis and advisory capabilities for materialized views, based on schema characteristics and previous workload history. These functions and procedures help users select from among the many materialized views that are possible in their schema.

In particular, the summary advisor can be used to estimate the size of a materialized view, recommend materialized views based on collected or hypothetical workload information, and report actual utilization of materialized views based on collected workload.

Optimized PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. It combines the ease and flexibility of SQL with the procedural functionality of a structured programming language. PL/SQL code can be stored centrally in a database.

Using PL/SQL stored procedures increases performance and optimizes memory usage because:

- Network traffic between applications and the database is reduced.
- A procedure's compiled form is readily available in the database, so no compilation is required at execution time.
- Multiple users can share one copy of a procedure in memory.

Oracle Database 10g introduces significant performance improvements with PL/SQL.

The PL/SQL compiler has been rewritten and provides a framework for efficient and ongoing optimization of compute-intensive PL/SQL programs. The new compiler includes a more sophisticated code generator and a global code optimizer that improves the performance of most programs substantially. The result is improved performance, especially for computationally intensive PL/SQL programs,

with a performance gain of about 2 times over Oracle9*i* Database Release 2 for a pure PL/SQL program¹⁵.

Additionally, the size of the PL/SQL executable code and dynamic stack have been significantly reduced. These size reductions improve overall performance, scalability, and reliability because execution of PL/SQL programs puts less pressure on memory, thus improving the performance of the overall Oracle system.

Also new in Oracle Database 10g, to help manage performance, PL/SQL compile time warnings automatically identify classes of PL/SQL constructs that are legal but could lead to poor run-time performance.

Oracle Database 10g also removes some restrictions on the native execution of PL/SQL that existed in Oracle9i Database. Native execution of PL/SQL programs provides the ability to compile PL/SQL modules to native code and presents several performance advantages: first, it eliminates the overhead associated with interpreting the byte-code and secondly, control flow and exception handling are much faster in native code than in interpreted code. As a result, execution speed of PL/SQL programs is greatly improved.

MANAGING LARGE VOLUMES OF DATA

Database installations managing huge volumes of data are common within Oracle's customer base. Database size is not an issue with Oracle Database: Oracle Database 10g can support extremely large databases, with up to 8 exabytes (8 million terabytes) of data.

Oracle Database includes powerful mechanisms that help create, deploy, manage, and use these huge amounts of data while at the same time dramatically improving performance for all types of database operations.

Support for Parallel Execution

Parallel processing distributes the work necessary to execute one or several tasks among multiple processors and I/O resources working simultaneously, allowing systems to scale by making optimal use of additional hardware resources.

When Oracle Database executes SQL statements in parallel, multiple processes work together simultaneously to execute a single SQL statement. By dividing the work necessary to execute a statement among multiple processes, Oracle can execute the statement more quickly than if only a single process executed it.

Oracle Database supports parallel processing between multiple processors of single systems and between multiple nodes of a cluster with Real Application Clusters.

Oracle Database supports parallel execution for all types of operations, including:

Performance with Oracle Database 10G

Page

¹⁵ See http://otn.oracle.com/tech/pl_sql/htdocs/new_in_10gr1.htm for more details

- Administrative operations: data loading, creations of tables and indexes, creation of temporary tables, partition maintenance, backup and recovery, replication.
- End-user operations: Queries, sub-queries in SELECT statements, INSERT, UPDATE, DELETE, OLAP analysis.

To guarantee optimal throughput and continuous, optimized system usage at any point in time, Oracle Database provides a parallel execution architecture that not only provides industry-leading performance but also is uniquely adaptive and dynamic.

Every Oracle Database instance provides a pool of parallel execution servers. The total number of available parallel execution servers is dynamically and automatically adjusted by the database system. If the number of parallel operations processed concurrently by an instance changes significantly, Oracle Database automatically changes the number of parallel execution servers in the pool.

The number of parallel execution servers that are used for an operation is the degree of parallelism (DOP). The DOP is determined at query execution time, based on the available hardware resources (in order to not overload the system), the actual workload of the system, the target object's parallel property, and the business priority of the request. For every operation, Oracle Database will automatically and dynamically adjust the degree of parallelism based on these factors, making optimal usage of the system's resources.

Parallel single cursor

In Oracle Database 10g, the parallel execution model for queries has moved from the slave SQL model to the parallel single cursor model. A single cursor contains all the information needed for parallel execution and is used for the entire parallel execution process.

The parallel single cursor architecture yields immediate benefits in terms of performance gains due to the parallel execution of operations that used to run in serial and the reduction of shared memory consumption.

Data Partitioning

Partitioning allows large database structures (tables, indexes, etc.) to be decomposed into smaller and more manageable pieces. Oracle Database offers several table partitioning methods designed to handle different application scenarios¹⁶:

http://otn.oracle.com/products/oracle9i/pdf/o9i_partitioning.pdf

¹⁶ For more information about Oracle's partitioning options, see Oracle9i partitioning, an Oracle white paper, May 2001

- Range partitioning uses ranges of column values to map rows to partitions.
 Partitioning by range is particularly well suited for historical databases.
 Range partitioning is also the ideal partitioning method to support 'rolling window' operations in a data warehouse.
- Hash partitioning uses a hash function on the partitioning columns to stripe data into partitions. Hash partitioning is an effective means of evenly distributing data.
- List partitioning allows users to have explicit control over how rows map to
 partitions. This is done by specifying a list of discrete values for the
 partitioning column in the description for each partition.
- In addition, Oracle Database supports range-hash and range-list composite partitioning.

Oracle Database also provides three types of partitioned indexes:

- A local index is an index on a partitioned table that is partitioned using the
 exact same partition strategy as the underlying partitioned table. Each
 partition of a local index corresponds to one and only one partition of the
 underlying table.
- A global partitioned index is an index on a partitioned or non-partitioned table that is partitioned using a different partitioning key than the table.
- A global non-partitioned index is essentially identical to an index on a non-partitioned table. The index structure is not partitioned.

Oracle Database allows all possible combinations of partitioned and nonpartitioned indexes and tables: a partitioned table can have partitioned and nonpartitioned indexes, and a non-partitioned table can have partitioned and nonpartitioned indexes.

Rolling window operations

Rolling window operations are very common in data warehousing environments. With this process, a data warehouse is periodically kept up to date by loading new data and purging old data in order to always keep the most recent data online. Range partitioning and local indexes make rolling window operations very efficient.

In the example illustrated below, a data warehouse maintains a window of one year of data, range partitioned by month. Loading a new month of data and dropping the oldest month does not affect the rest of partitions. The time to complete the entire operation is directly proportional to the amount of new data and independent of the size of the data warehouse.

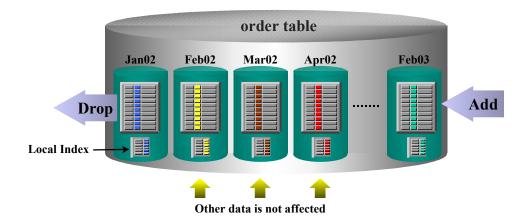


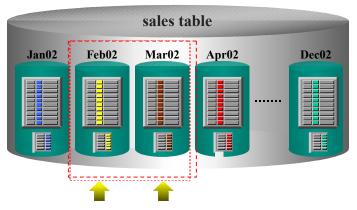
Figure 6: Rolling Window operation

Partition Pruning

Partition pruning enables operations to be performed only on those partitions containing the data that is needed. The query optimizer analyzes FROM and WHERE clauses in SQL statements to eliminate the unneeded partitions: partitions that do not contain any data required by the operation are eliminated from the search. This technique dramatically reduces the amount of data retrieved from disk and shortens the use of processing time, improving query performance and resource utilization. In the example shown in figure 6 the Sales table for the year 2002 has been partitioned by month. After analyzing the WHERE clause of the query, the optimizer determines that only the 2 partitions corresponding to the months of February and March need to be accessed during query execution. The query could potentially execute 6 times faster simply because of partition pruning.

Oracle Database fully implements partition pruning, including with composite partitioned objects. In addition, partition pruning can be combined with index access (global or local). When the index and the table are partitioned on different columns, Oracle Database will eliminate unneeded index partitions, even when the partitions of the underlying table cannot be eliminated.

SELECT sum(sales_amount)
FROM sales
WHERE sales_date
BETWEEN '01-Feb-2002'
AND '31-MAR-2002';



Only 2 partitions are accessed

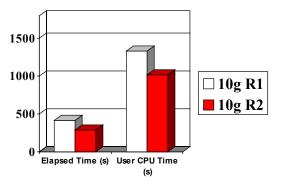


Figure 8: Partition pruning improvements

Figure 7: Partition pruning: example

By supporting a wide variety of partitioning options and combinations for tables as well as for indexes, Oracle Database enables applications to take advantage of partition pruning in more business scenarios.

Partition pruning techniques have been further improved with Oracle Database 10g Release 2. With better pruning optimizations and decreased number of physical I/O requests, the performance of some queries¹⁷ can improve significantly, as can be seen in figures 8 and 9.

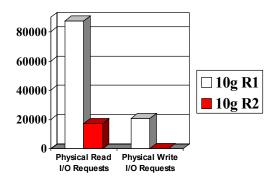


Figure 9: Partition pruning improvements, IO requests

Partition-Wise Joins

With Oracle Database, partitioning can also improve the performance of multitable joins, by using a technique known as partition-wise joins.

Partition-wise joins can be full or partial. Full partition-wise joins can be applied when two tables are being joined together, and both of these tables are partitioned on the join key. Partial partition-wise joins can be applied when only one of the tables is partitioned on the join key. In this case, Oracle Database dynamically redistributes usually the smaller non-partitioned table based on the partitioning key of the larger, partitioned reference table. Once the other table is dynamically repartitioned, the execution is similar to a full partition-wise join.

¹⁷ For further details on the tests results for Oracle Database 10g Release 2, see DSS performance in Oracle Database 10g Release 2, an Oracle white paper, May 2005.

Because partition-wise joins break a large join into smaller joins that occur between each of the partitions, the overall join is completed in less time. This offers significant performance benefits both for serial and parallel execution.

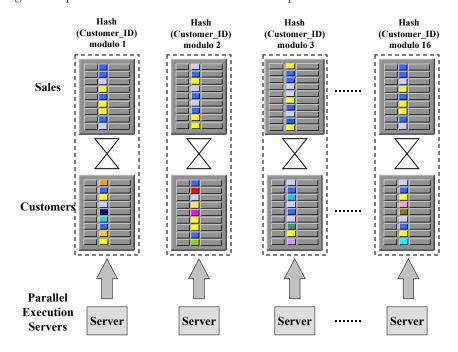


Figure 10: Full partition-wise join

Partition-wise joins also improve query response time by minimizing the amount of data exchanged among parallel execution servers when joins execute in parallel. This significantly reduces response time and makes more efficient use of both CPU and memory resources.

Global indexes

Global indexes are commonly used for on-line transaction processing (OLTP) environments, in which the ability to efficiently access any individual record using different criteria is one of the fundamental requirements. For this reason, most OLTP systems have many indexes on large tables. Oracle's own E-Business suite of applications has a dozen or more indexes on many of its large tables. Global partitioned indexes are more flexible in that the degree of partitioning and the partitioning key are independent of the table's partitioning method. This is illustrated in the following figure where the Customers table is partitioned on the Customer_ID key and a global index can be created and partitioned on the Customer_Name key.

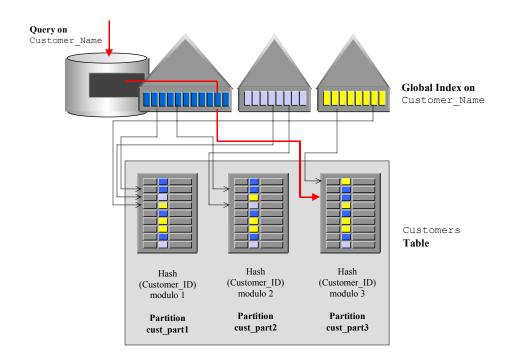


Figure 11: Example of global partitioned index

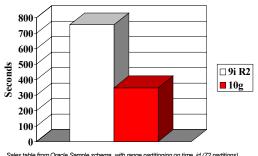
With Oracle Database 10g users can now hash-partition indexes on tables, partitioned tables, and index-organized tables. This provides increased throughput for applications with large numbers of concurrent inserts.

Large number of partitions

The increasingly popular usage of partitioning and the continuously growing size of data warehouses have made possible the creation of database structures with tens of thousands partitions. With Oracle Dabatase 10g Release 2, the maximum number of partitions per table increases from 64K to 1024K, providing more flexibility and scalability for databases containing very large tables.

Oracle Database 10g, also dramatically improves the scalability and memory usage of partitioned objects to make sure such numbers have a limited performance impact on the operations made on these objects.

As an example, the performance improvement for the DROP TABLE operation on a table with 18,432 partitions shows a reduction of the elapsed time of about 56% from Oracle9*i* Database to Oracle Database 10*g*¹⁸.



Sales table from Oracle Sample schema, with range partitioning on time_id (72 partitions) and hash partitioning on prod_id (256 sub-partitions per partition) \sim 72 x 256 = 18.432 partitions

Figure 12: DROP TABLE with large number of partitions

¹⁸ See "DSS Performance in Oracle Database 10g", an Oracle white paper, September 2003

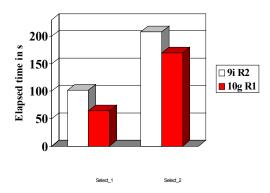


Figure 13: Full table scans

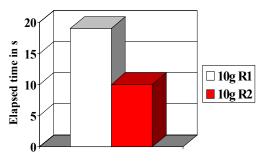


Figure 14: In-memory Sort

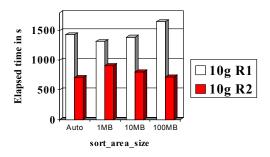


Figure 15: In-memory Sorts

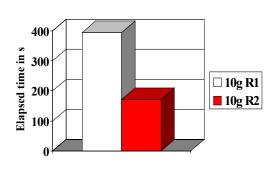


Figure 16: hash-based aggregation

Full Table Scans

Performance of full table scans has been significantly improved with Oracle Database 10g. Improvement factors in elapsed time and CPU utilization are between 40 and 60% for single table scans with simple predicates or no predicates.

The figure to the left illustrates the improvement of performance measured for full table scans of a compressed table¹⁹.

In-memory Sorts

Oracle Database 10g Release 2 introduces a new in-memory sort algorithm which sorts faster and has better memory system locality than the existing sort algorithm in Oracle Database 10g Release 1. The new in-memory sort algorithm improves performances²⁰ for a variety of queries and operations: queries with an "order by" clause (see figure 14), queries using "sort merge join", queries using "partition outer join", create index.

The new algorithm significantly improves query performance in terms of elapsed time while consuming less CPU and allowing more concurrent queries. Because of the better memory system locality afforded by the new in-memory sort algorithm, the performances of queries using the new in-memory sort algorithm improve as the available memory increases (see figure 15 for CREATE INDEX with different sort_area_sizes values).

Hash-based Aggregation

Data aggregation is a frequent operation in data warehousing and OLAP environments. Data is either aggregated on the fly or pre-computed and stored as materialized views. In either case, the time and resources needed to do the aggregation must be minimal. Oracle Database 10g Release 2 improves aggregation performance by using a hash-based method for aggregation replacing the sort-based approach used in previous releases. The hash-based scheme significantly improves the performance (see figure 16) of data aggregation²¹.

High-speed Data Movements

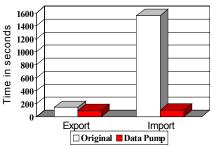
Oracle Database 10g provides new capabilities to extract, load, and transform data in order to facilitate the efficient building and refreshing of large data warehouses or multiple data marts.

For bulk movement of data, Oracle Database 10g provides cross platform support for transportable tables, allowing large amounts of data to be very quickly detached from a database on one platform, and then re-attached to a database on a different platform.

¹⁹ See "DSS Performance in Oracle Database 10g", an Oracle white paper, September 2003

²⁰ For further details on the tests results for Oracle Database 10g Release 2, see DSS performance in Oracle Database 10g Release 2, an Oracle white paper, May 2005.

²¹ see footnote 20



1.0 GB data (9.3 M rows) from large portal company, single stream Hardware: 1-CPU Ultra 60, 1 GB memory, 2 disk drives

Figure 17: Data Pump Import/Export Utilities

Oracle Database 10g also provides new Data Pump utilities. Oracle Data Pump is a high-speed, parallel infrastructure that enables quick movement of data and metadata from one database to another. This technology is the basis for Oracle's new data movement utilities, Data Pump Export and Data Pump Import.

The design of Data Pump Export and Import results in greatly enhanced performance over original Export and Import. The graph in figure 17 compares times elapsed for single stream data movements using the original export and import utilities and the new Data Pump utilities, respectively:

Moreover, by using the PARALLEL parameter, the maximum number of threads of active execution servers operating on behalf of the Import or Export job can be specified, resulting in even better performance in unloading and loading data.

The new Data Pump Export and Import utilities provide much more flexibility in object selection: there is support for fine-grained object selection, based upon objects and object types. The new utilities also support a "network" mode, allowing for a file-less, overlapping Export/Import operation to occur between source and target databases. A PL/SQL package allows users to write their own data movement utilities using publicly documented interfaces.

Management of the Export/Import environment has also been improved in this release. Data Pump operations can be completely re-started regardless of whether the job was stopped voluntarily by the user or something unforeseen terminates the job. Users have the ability to determine how much disk space will be consumed for a given job and be able to estimate how long it will take to complete. Administrators can monitor jobs from multiple locations by detaching from and reattaching to long-running jobs, and can modify certain attributes of the job, such as parallelism and dumpfile availability.

SUMMARY OF ORACLE DATABASE 10g's NEW FEATURES FOR **PERFORMANCE**

(Release 2 features in bold)

Area	Feature
Table structures	Sorted hash clusters
	Full table scan
Partitioning	Global hash indexes
	List partitioning support for Index-Organized Tables
	Improved performance for operations on large number of partitions
	Support for 1 million partitions per table
	Improved partition pruning techniques
Windows-based systems	Support for 64-bit Windows
Networking	Support for High-speed Infiniband Network
Materialized Views	Improved partition-aware refresh
	Query rewrite can use more than one materialized view
Cost-based optimizer	Automatic statistics collection
	CPU+IO cost model
	Automatic Tuning Optimizer
Transactions	Asynchronous commit
Complex queries	In-memory Sorts
	Hash-based Aggregations
PL/SQL	New code generator and global optimizer
Self-tuning Memory	Self-tuning SGA
	SGA size advisor
	Self-tuned checkpoints
	Object checkpointing
	Self-tuning multi-block read count
ETL	Data Pump Export and Import Utilities
	Cross-platform transportable tablespaces



Database Performance with Oracle Database 10g

December 2003 Author: Hervé Lejeune

Contributing Authors: Vineet Buch, Carol Palmer

Oracle Corporation World Headquarters 500 Oracle Parkway Redwood Shores, CA 94065 U.S.A.

Worldwide Inquiries: Phone: +1.650.506.7000 Fax: +1.650.506.7200 www.oracle.com

Oracle Corporation provides the software that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various product and service names referenced herein may be trademarks of Oracle Corporation. All other product and service names mentioned may be trademarks of their respective owners.

Copyright © 2000 Oracle Corporation All rights reserved.