Technical Comparison of Oracle Database vs. SQL Server 2000: Focus on Performance

An Oracle White Paper
May 2005

ORACLE

Technical Comparison of Oracle Database vs. SQL Server 2000: Focus on Performance

Introduction 3
Platform availability 4
Concurrency Model 5
  Multi-Version Read Consistency 5
  Non-Escalating Row-Level Locking 7
  Impact of locking on packaged applications 8
Indexing capabilities 8
  Bitmap Indexes & Bitmap Join Indexes 8
Partitioning 9
  Oracle 's Partitioning Options 9
  SQL Server 2000's Partitioning Options 10
  Performance benefits of partitioning 11
    Global indexes 11
    Partition Pruning 11
    Partition-Wise Joins 12
Parallel Execution of Operations 12
Clustering 13
  Oracle Real Application Clusters 13
  SQL Server 2000 Federated Databases Architecture 14
    Inapplicability to Real-World Applications 15
    Inability to Scale 16
    Limited Processing Power Working on One Partition 18
Additional Data Warehousing capabilities 18
  Merge 18
  Multi-Table Inserts 19
  Pipelined Table Functions 20
Conclusion 20

Technical Comparison of Oracle Database vs. SQL Server 2000: Focus on Performance

INTRODUCTION

Performance is a requirement for all databases, but the requirements for that performance vary greatly, depending on the types of applications accessing the database

Online transaction processing (OLTP) and e-commerce applications are characterized by increasingly large user populations concurrently accessing large volumes of data for short and frequent insert or update transactions. Such environments require support for high throughput, a variety of available indexing strategies, and excellent data concurrency.

Data warehouses are very large databases specifically designed for periodically loading massive amounts of data for frequent and ad-hoc complex queries. A data warehouse has constantly growing data volumes and numbers of accessing applications and users. It should be optimized to perform well for a wide variety of long-running, ad-hoc queries. Such environments require adequate support for query rewrite capabilities, efficient indexing capabilities, a wide selection of partitioning strategies, and extended support for parallel execution.

Oracle Database outperforms SQL Server 2000 on a wide variety of industry and ISV-specific benchmarks, and is recognized as the industry leader in database scalability.

This document describes the technical differences between Oracle Database 10*g* Release 2, the latest version of Oracle Database, and SQL Server 2000 that explain Oracle Database's superior performance in both benchmarks as well as real customer environments. After a short introduction to the differences in platform availability, the document focuses on the major techniques commonly used to ensure good performance and scalability in modern, enterprise-class, relational database systems: concurrency model, indexing, partitioning, parallel execution, and clustering.

Additionally, it gives an introduction to some of the unique features that Oracle Database provides to enhance the performance of ETL processes in Data Warehousing environments.

The main differentiators between the two products are summarized in the table below:

Feature	Oracle Database	SQL Server 2000[1]
Concurrency Model	Multi-version read consistency Non-Escalating row-level locking	Shared read locks or dirty reads Locks escalate
Indexing capabilities	B-Tree indexes Index-organized Tables Bitmap indexes Bitmap Join Indexes	B-Tree indexes Clustered Indexes Not supported Not supported
Partitioning options	Range, hash, list and composite partitioning Local and global indexes	Not supported Only local indexes with member tables
Parallel execution	Queries, INSERT, UPDATE, DELETE	Queries only
Clustered configurations	Transparent scalability with Real Application Clusters	Requires data partitioning in member tables and Distributed Partitioned Views
Additional data warehousing capabilities	Materialized Views MERGE Multi-table INSERT Pipelined table Functions	Indexed Views Not supported Not supported Not supported

Table 1: Key Differentiators

PLATFORM AVAILABILITY

Even though product portability is not directly linked to performance, software availability across a wide variety of hardware and operating systems enables

[1] From SQL Server 2000 documentation, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/startsql/getstart_4fht.asp

users to seamlessly upgrade or replace their hardware systems without having to worry about changing, redesigning or rebuilding their applications. In other words, cross-platform availability helps preserve the initial investments in application software and helps deliver performance consistency across multiple platforms.

Oracle Database is available on a large selection of hardware and operating systems, scaling from low-end uni-processor servers to large symetrical multi-processor machines to multi-node clusters. Oracle Database supports all major Unix platforms, including Linux, Microsoft operating systems, and a variety of other systems, including OS/390 mainframes. With Oracle, users are able to upgrade hardware and operating systems without changing or rewriting their applications.

SQL Server 2000 only runs on Microsoft's operating systems. Customers wishing to upgrade hardware are limited to platforms running these systems and must face the cost of converting their systems completely if they ever outgrow the capacity of their platform.

CONCURRENCY MODEL

In multi-user environments, concurrency control ensures that data updates made by one user do not adversely affect those made by other users. Oracle and SQL Server 2000 differ greatly in their implementation of concurrency control. The main differences are summarized in the table below and further explained in the following sections.

Oracle Database	SQL Server 2000
Multi-version read consistency	Not available
No read locks	Requires shared read locks to avoid dirty reads
No dirty reads	Dirty reads if not using shared locks
Non-escalating row-level locking	Locks escalate
Readers don't block writers	Readers block writers
Writers don't block readers	Writers block readers
Minimal deadlocks under load	Deadlocks can be a serious problem under load

Table 2: Concurrency Models

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 that satisfy 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's implementation of multi-version read consistency always provides consistent and accurate results. When an update occurs in a transaction, the original data values are recorded in the database's undo records. Oracle uses the current information in the undo records 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. Other databases, such as SQL Server 2000, have to choose a workaround to avoid concurrency problems, such as locking data to prevent it from changing while being read or preventing or queries from reading changed but uncommitted information.

SQL Server 2000 does not provide multi-version read consistency. Instead it requires applications to either use shared locks for read operations, with various levels of isolation, or to accept dirty reads. Shared locks prevent data that is read from being changed by concurrent transactions. Clearly, this implementation restricts the ability of the system to properly service concurrent requests in environments involving a mix of reads and writes, as explained in Microsoft's documentation: *"SQL Server, in contrast, uses shared locks to ensure that data readers only see committed data. These readers take and release shared locks as they read data. These shared locks do not affect other readers. A reader waits for a writer to commit the changes before reading a record. A reader holding shared locks also blocks a writer trying to update the same data."*[2]

A consequence is that *"releasing locks quickly for applications that support high numbers of users is more important in SQL Server than in Oracle."*[3]

The only alternative developers have to t his problem is to build separate workload environments, where intensive read activities, such as reporting, cannot interfere with on-line transactional applications. Regardless of which

2 See "Migrating Oracle Databases to SQL Server 2000", SQL Server 2000 Resource kit, Chapter 7, Locking and Transaction Isolation

3 "Migrating Oracle Databases to SQL Server 2000", SQL Server 2000 Resource kit, Chapter 7, Locking and Transaction Isolation

approach is used, SQL Server 2000 developers usually have to find some compromise in their application design in order to get acceptable data concurrency and accuracy.

In Oracle, writers and readers never block each other. Oracle's powerful multi-version read consistency allows mixed workload environments to function properly without incurring any performance penalty for the users.

Non-Escalating Row-Level Locking

Row-level locks offer the finest granularity of lock management, and thus, 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 uses row-level locking as the default concurrency model and stores locking information within the actual rows themselves. By doing so, Oracle can have as many row level locks as there are rows or index entries in the database, providing unlimited data concurrency.

"In practice and under high load, SQL Server's locking system, which is based on lock escalation, does not perform well. Why? Lock contention.

...

Granted, if you've only got a few occasional users, you won't have much trouble with SQL Server's out-of-the-box behavior. You'll be hard pressed to see these problems with simple in-the-office tests or deployments involving just a few users. But throw a couple hundred concurrent users at your database and a constant stream of INSERTS and UPDATES with quite a few DELETEs sprinkled in, and you'll start reading Oracle literature and eyeing your war chest."[4]

SQL Server 2000 also supports row-level locking as the default concurrency model. However, because it was not the default level of lock granularity in earlier versions of the database, the late addition of row-level locking was made possible only through the use of additional, separate pools of lock structures.

As with any memory structure, these lock structures are limited in their size and thus limit the maximum number of locks that can be supported by the database. The maximum amount of memory devoted to locks is limited to 40% of the amount reserved for the database. SQL Server dynamically determines the appropriate level at which to place locks for each statement but the level at which locks are acquired does not depend on the memory available: a small table may have a table lock applied, while a larger table may have row locking applied. There is a lock escalation threshold, even though it is automatically adjusted. Since SQL Server uses many more locks because of read locks, reaching a threshold value is not uncommon. A consequence is that as more users access the application and transaction volume increases, SQL Server 2000 will escalate row level locks to table locks to conserve memory. This in turn means that fewer users can access the data at the same time – users will have to wait.

"Locking at a smaller granularity, such as rows, increases concurrency, but has a higher overhead because more locks must be held if many rows are locked. Locking at a larger granularity, such as tables, are expensive in terms of concurrency because locking an entire table restricts access to any part of the

[4] http://www.sql-server-performance.com/lock_contention_tamed_article.asp

table by other transactions, but has a lower overhead because fewer locks are being maintained."[5]

While the result of this lock escalation is that the total number of locks being held is reduced, the likelihood of having two or more users waiting for data locked by each other is greatly increased. These unpleasant deadlock situations can result in aborting one or more of the concurrent users' transactions.

Oracle never escalates locks and, as a consequence, 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 code, as described above.

More importantly, packaged applications frequently cannot be easily changed or maintained to support less than optimal locking schemes. Oracle's powerful multi-version read consistency model provides scalability benefits to all applications, which is a major reason that a majority of packaged applications run on Oracle, including SAP R/3, PeopleSoft, Siebel and Oracle Applications.

INDEXING CAPABILITIES

Indexes are database structures that are created to provide a faster path to data. Using indexes can dramatically reduce disk I/O operations, thus increasing the performance of data retrieval.

Both Oracle and SQL Server 2000 support traditional B-Tree indexing schemes, which are ordered lists of key values, associated with the storage location of the table row that contain these values.

Both also support index-organized tables, which are called clustered indexes in Microsoft's terminology. Index-organized tables provide fast access to table data for queries involving exact match and/or range search on the primary key because table rows are stored in the leaf nodes of the primary key index.

However, Oracle also supports static bitmap indexes and bitmap join indexes, whose usage can provide huge performance benefits for typical load and query operations in data warehousing environments. These indexing schemes are explained further 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 (ROWIDs). Each bit in the bitmap

---

[5] from Microsoft SQL Server documentation: Understanding Locking in SQL Server
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_7a_7xde.asp

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 ROWIDs, especially for low cardinality data. 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 dramatically[6].

With Oracle, it is also possible to create bitmap indexes on index-organized tables, thereby allowing index-organized tables to be used as fact tables in data warehousing environments.

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 bit-wise 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.

SQL Server 2000 does not support bitmap indexes and bitmap join indexes.

PARTITIONING

Partitioning allows large database structures (tables, indexes, etc.) to be decomposed into smaller and more manageable pieces. Although it is primarily considered a feature for manageability and availability, partitioning also provides a number of performance benefits.

Oracle 's Partitioning Options

Oracle Database offers several table partitioning methods designed to handle different application scenarios[7]:

- Range partitioning uses ranges of column values to map rows to partitions. Partitioning by range is particularly well suited for historical

[6] See Key Data Warehousing Features in Oracle Database 10g: A Comparative Performance Analysis, An Oracle White Paper, April 2005
http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_bi_dw_key_features_10gr2_0405.pdf

[7] For more information about Oracle 's partitioning options, see Partitioning in Oracle Database 10g, an Oracle white paper, February 2005
http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_dss_partitioning_10gr1_0205.pdf

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 supports range-hash and range-list composite partitioning.

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

SQL Server 2000's Partitioning Options

SQL Server 2000 does not support partitioning as generally defined in the database industry. Instead, it supports partitioned views.

A partitioned view joins horizontally partitioned data from a set of member tables across one or more servers, making the data appear as if from one table. The data is partitioned between the member tables based on ranges of data values in one of the columns, called the partitioning column. The data ranges for each member table are defined in a CHECK constraint specified on the partitioning column. The view uses UNION ALL to combine selects of all the member tables into a single result set.

SQL Server 2000 distinguishes between local and distributed partitioned views. In a local partitioned view, all participating tables and the view reside on the same instance of SQL Server. In a distributed partitioned view, at least one of the participating tables resides on a different (remote) server. Distributed Partitioned Views are further discussed in the Clustering section of this document.

Performance benefits of partitioning

Due to its lack of real partitioning capabilities, SQL Server 2000 suffers from many important deficiencies in terms of performance and scalability.

Global indexes

Global indexes are commonly used for OLTP environments and offer efficient access to any individual record. Global partitioned indexes are more flexible in that the degree of partitioning and the partitioning key are independent from the table's partitioning method. This is illustrated in the following figure where the `Employees` table is partitioned on the `deptno` key and a global index can be created and partitioned on the `empno` key.

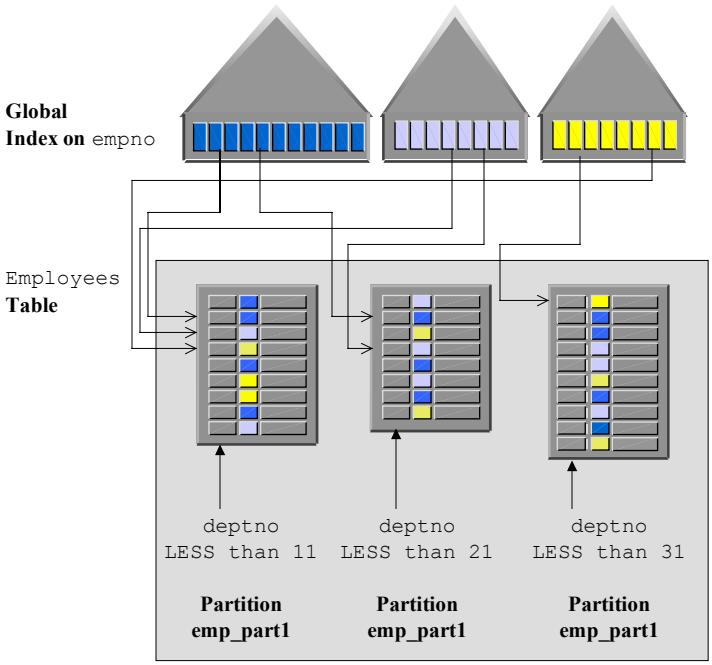

Figure 1: Oracle's Global Partitioned Indexes

SQL Server 2000's partitioned views do not support global indexes. Queries that do not specify a search condition on the "partition" column will have to search all the tables. This behavior results in even worse performance results in the case of clustered environments where member tables reside on separate nodes (see the section on Clustering). This lack of support considerably restricts the ability of SQL Server 2000 to be used in real-world OLTP applications.

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. It is an essential performance feature for data warehousing environments.

Oracle 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 will eliminate unneeded index partitions, even when the partitions of the underlying table cannot be eliminated.

SQL Server 2000 implements rudimentary partition pruning with local partitioned views, when all participating tables and the view reside on the same instance of SQL Server. If CHECK constraints are used on the partition columns when creating the member tables, the query optimizer will determine which member tables contains the rows and will limit the search to these tables. However, overall performance effectiveness is limited since this technique cannot be combined with global indexes and composite partitioning, which are not supported.

Partition pruning with Distributed Partitioned Views is severely restricted and suffers from lack of scalability. This is explained in detail in the Clustering section of this document.

Partition-Wise Joins

With Oracle, partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise joins. Partition-wise joins can be applied when two tables are being joined together, and both of these tables are partitioned on the join key. Partition-wise joins break a large join into smaller joins that occur between each of the partitions, completing the overall join in less time. This offers significant performance benefits both for serial and parallel execution. SQL Server 2000 does not support partition-wise joins.

PARALLEL EXECUTION OF OPERATIONS

Parallel execution of SQL operations can vastly improve the performance for operations involving large volumes of data. It helps reduce response time for data-intensive operations on large databases typically associated with decision support systems and data warehouses.

Oracle will execute INSERT, UPDATE, DELETE, and MERGE[8] statements in parallel when accessing both partitioned and non-partitioned database objects.

With SQL Server 2000, INSERT, UPDATE, and DELETE statements are executed serially (MERGE is not supported).

---

[8] The MERGE statement is explained later in this document, in the section on Additional Data Warehousing capabilities

CLUSTERING

Clusters are groups of independent servers, or nodes, connected via a private network (called a cluster interconnect), that work collaboratively as a single system. Clusters allow applications to scale beyond the limits imposed by single node systems when processing loads exceed the capacity of large individual servers.

Only Oracle provides real support for clustered configurations, where full and transparent scalability can be obtained by simply adding new nodes as the demand increases. As indicated in Microsoft's documentation, "*SQL Server 2000 does not support this type of clustering"*[9]. Instead, users are forced to use a federation of databases to achieve some kind of scalability. The following sections explain in more details how both architectures differ and the impact on performance and scalability.

Oracle Real Application Clusters

Real Application Clusters (RAC) is the Oracle Database option that supports hardware clusters.

Oracle Real Application Clusters uses a shared disk approach. In a shared disk database architecture, database files are logically shared among the nodes of a loosely coupled system with each instance having access to all the data.

Oracle Real Application Clusters uses the patented Cache Fusion™ architecture, a technology that utilizes the interconnected caches of all the nodes in the cluster to satisfy database requests for any type of application (OLTP, DSS, packaged applications). Query requests can now be satisfied both by the local cache as well as any of the other caches. Update operations do not require successive disk write and read operations for synchronization since the local node can obtain the needed block directly from any of the other cluster node's database caches. Oracle Cache Fusion™ exploits low latency cluster interconnect protocols to directly ship needed data blocks from the remote node's cache to the local cache. This removes slow disk operations from the critical path of inter-node synchronization. Expensive disk accesses are only performed when none of the caches contain the necessary data and when an update transaction is committed, requiring disk write guarantees. This implementation effectively expands the working set of the database cache and reduces disk I/O operations to dramatically speed up database operations.

The utilization of Cache Fusion in Oracle allows customers to easily take advantage of the scalability provided by Oracle Real Application Clusters, with little or no performance cost. Customers can horizontally scale the database tier as demand grows. Because the full Cache Fusion implementation in Oracle

---

[9] From SQL Server 2000 documentation, in SQL Server Architecture, Federated SQL Server 2000 Servers
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8_ar_cs_4fw3.asp

eliminates the latencies associated with disk based cache coordination, applications can now scale effectively without having to be cluster aware.

Additionally, cluster load balancing features enable a large number of user connections to be automatically and transparently distributed among cluster nodes, making optimal use of the overall cluster processing capabilities.

SQL Server 2000 Federated Databases Architecture

Distributed partitioned views can be used to implement a federation of database servers[10]. A federation is a group of servers administered independently, but which cooperate to share the processing load of a system.

With a federated database, the data is divided between the different servers. When a user normally connects to a federated database, they are connected to one server. If the user requests data that happens to reside on a different server, the retrieval takes significantly longer than retrieving data stored on the local server.

Microsoft's documentation recommends avoiding remote execution of queries:

"If you have an extremely high-hit database, it may not be that database or table that you want to distribute.

...

It is recommended that most of the SQL statements be routed directly to the member server with a large percent of the necessary data, therefore minimizing the distributed nature of the design.

...

When distributing data across multiple servers, there is a performance hit for querying a remote server. Analysis should be done on the type of queries implemented in the OLTP environment to get a baseline on what data is being touched by specific queries. Because there is a certain degree of overhead in running distributed queries, this overhead may in some cases outweigh the benefits of distributing your tables."[11]

This can be an impossible task, especially with large proprietary applications. This is admitted in one of the books written on SQL Server 2000:

"This feature set permits partitioning of views to horizontally partition tables across multiple servers. This feature enabled SQL Server 2000 to set its new Transaction Processing Performance Council (TPC) TPC-C benchmark of 262,243 transactions per minute (tpmC) with 12 clustered Compaq systems.

---

[10] See Database Architecture: Federated vs. Clustered, An Oracle White Paper, March 2002, http://otn.oracle.com/deploy/performance/pdf/FederatedvsClustered.pdf

[11] From SQL Server 2000 documentation, Distributed Partitioned Views Recommendations

Unfortunately, the current design has significant limitations and it's not easy to implement.[12]

Because of this, developers and administrators must build their systems to avoid this occurrence. To build a federation of database servers, administrators and developers must[13]:

1. *Create multiple databases, each on a different member server running an instance of SQL Server 2000.*
2. *Partition the individual tables in the original database so that most related data is placed together on a member server. This may require different methods of distributing the data in the various tables across all the member databases; partitioning some tables; making complete copies of other tables in each member database; and leaving some tables intact on the original server.*
3. *Devise data routing rules that can be incorporated in the business services tier, so that applications can send each SQL statement to the member server that stores most of the data required by the statement.*

Obviously this implementation process requires an excellent knowledge and a deep understanding of both the application logic and the database layout. When scaling out an existing application, database administrators need to identify which tables are good candidates for being partitioned and which tables should be replicated on all servers. This process assumes that the partition keys on which the partitions are based were correctly selected. Developers then may have to redesign large parts of the business logic of these applications to make sure that each SQL statement is routed to a member server that contains most, if not all, of the data required to process the statement.

The end result of these workarounds means that SQL Server 2000's federated database design can only be used for certain applications.

[Distributed partitioned Views], … can be helpful in the case of data warehousing applications, but with the current functionality it leaves a lot to be desired for making a successful implementation for the OLTP applications."[14]

The difference in the architecture adopted in the two products has many consequences in terms of performance and scalability.

Inapplicability to Real-World Applications

Popular OLTP application suites, such as the Oracle eBusiness Suite, SAP, or PeopleSoft, have thousands of tables. As we have seen earlier, with SQL Server

---

[12] p.42, Microsoft SQL Server 2000, A Guide to Enhancements and New Features, Rahul Sharma, Addison Wesley

[13] From SQL Server 2000 documentation: SQL Server Architecture, Partitioning data

[14] p.232, Microsoft SQL Server 2000, A Guide to Enhancements and New Features, Rahul Sharma, Addison Wesley

2000's federated databases all tables must be partitioned or replicated on all nodes to ensure some sort of scalability. Porting these applications to such an environment would be a very complex and expensive process.

Microsoft itself admits the fact that the federated databases architecture is not an easy solution to chose when considering scaling out applications:

"The implementation of distributed partitioned views can bring about a lot of complexity to the management and operation of the overall environment. Currently, the percentage of companies that may need to implement this scale-out behavior to improve their environment is very small.[15]*"*

In contrast, Oracle Real Application Clusters provide full application compatibility. All types of applications can scale effectively without having to be specifically tailored for clustering environments. Migration is transparent: no specific redesign or code changes are required for existing applications, no explicit application segmentation or data partitioning is required.

Inability to Scale

The fact that federated database servers are composed of independent databases, with no common data dictionary and no support for global indexes, imposes severe performance and scalability penalties on applications.

We will first consider the simple and very common case of a query referencing a partitioned view, and specifying a search condition on the partitioning column. If the member server to which the query is routed does not contain all the data required to process the statement, the query will have to be propagated to the other member servers. Even though the query processor will optimize this process by limiting this propagation to the member servers that contain data satisfying the request[16], there is a performance hit for querying these remote servers. The performance and scalability will degrade as the number of member servers involved in the query execution increases. As stated in Microsoft's documentation: "*because there is a certain degree of overhead in running distributed queries, this overhead may in some cases outweigh the benefits of distributing your tables*"[17].

This lack of scalability gets worse when the query referencing the partitioned view does not specify the partition key. This is also a very common case: the ability to efficiently find records using different criteria is one of the fundamental requirements for OLTP databases. For this reason, most OLTP systems have many indexes on large tables. Oracle's own eBusiness suite has a dozen or more indexes on many of its large tables.

Since SQL Server 2000 does not support global indexes, that is, indexes that span partition boundaries, such queries must be broadcast to all nodes that

---

[15] From SQL Server 2000 documentation, Distributed Partitioned View recommendations
[16] From SQL Server 2000 documentation, Resolving Distributed Partitioned Views
[17] From SQL Server 2000 documentation, Distributed Partitioned View recommendations

constitute the federated database. The more nodes there are in the federated system, the worse the impact on performance. In the example illustrated below, a federated database is built by partitioning a `Customer` table into four member tables residing on four different nodes. The partitioning is done on the `Customer_ID` key. A simple query based on the customer name will have to be propagated to the four member servers of the cluster, seriously impacting the overall performance of the query.

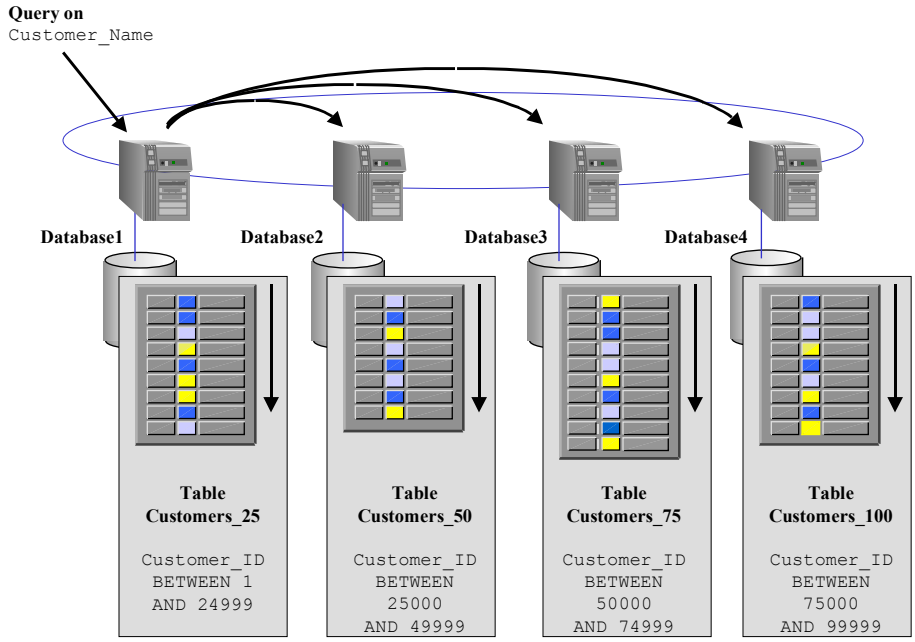

Figure 2: Query Broadcasting with SQL Server2000 Federated Databases

Oracle does not have any of these limitations.

All nodes in the cluster have equal access to the same, unique database: the node executing the query has full access to the table referenced in the SQL statement. There is no need to propagate the query to any other node in the cluster, and this is true in both cases mentioned above. Partition pruning will work transparently without restriction. If the `Customers` table is partitioned on the `Customer_ID` key, and the query is based on the same key, only partitions that contain data satisfying the query will be accessed.

And Oracle has full support for global indexes. A similar application to the second case described above running on Oracle Real Application Clusters would make use of the global index (partitioned or not) defined on the `Customer_Name` key to directly access the correct partition, without any performance penalty. This is illustrated in the figure below.

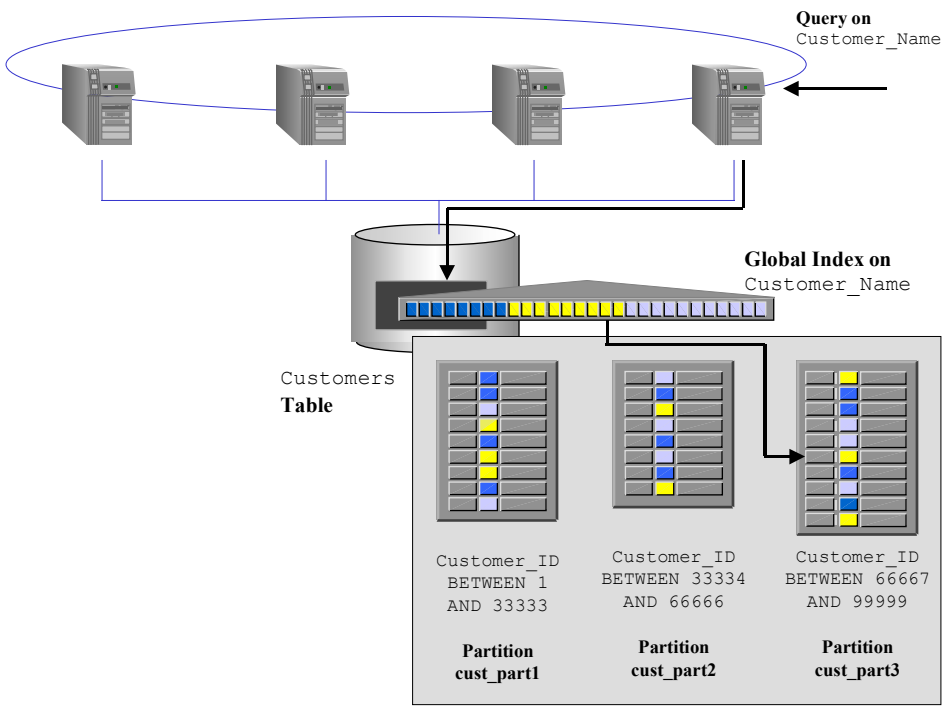

Figure 3: Query Execution with Oracle RAC

Limited Processing Power Working on One Partition

With SQL Server, only the node that owns the partition can contribute to reading that partition. The processing power will always be limited to the processing power of the node to which the table belongs.

Oracle does not have this limitation: it can potentially deploy the whole processing power of the system, i.e., all parallel execution servers, from all nodes, to process one particular partition if necessary.

ADDITIONAL DATA WAREHOUSING CAPABILITIES

Oracle provides several unique features useful in data warehousing environments, in particular during the Extraction, Transformation and Loading (ETL) process. During these phases, various methods are used to access and manipulate source data and load it into a data warehouse.

Merge

The MERGE statement is a new SQL statement that provides the ability to update or insert rows conditionally into a table or a view, depending upon which is needed, reducing the number of application statements and application complexity. This statement is a convenient way to combine at least two

operations, avoiding the need to use multiple INSERT and UPDATE DML statements.

The MERGE statement can be used to select rows from one table for update or insertion into another table. Such operations are frequently used in a number of data warehousing applications where tables need to be periodically refreshed with new data arriving from on-line systems. This new data might contain changes to the existing rows of the warehouse table or might introduce a new row altogether. The MERGE statement typically addresses these types of situations.

MERGE brings significant performance improvement due to the optimization of execution and the reduction of scan and join operations compared to what would be performed using an equivalent sequence of DML statements.[18]

SQL Server 2000 does not support an equivalent of the MERGE statement. Without MERGE, these operations can only be expressed as a sequence of INSERT and UPDATE statements. This approach suffers from deficiencies in both performance and usability.

Multi-Table Inserts

Multi-table INSERTs allow data to be inserted into more than one table using a single SQL statement, which is more efficient than using multiple, separate SQL statements for each table.

This feature is very useful in data warehousing systems, where data is transferred from one or more operational data sources to a set of target tables. Multi-table inserts extend the scope of the INSERT . . . SELECT statement to insert rows into multiple tables as part of a single DML statement.

This new feature brings significant performance improvement[19] due to optimization of execution and reduction of scan operations on the source data. No materialization in temporary tables is required and source data is scanned once for the entire operation instead of once for each target table with multiple statements.

Multi-table inserts make SQL more useful for data transformations and conditional handling and allow faster loading and transformations of large volumes of data.

SQL Server 2000 does not support multi-table inserts, meaning that similar operations can only be expressed as a sequence of INSERT statements, requiring more scan operations on the source data.

---

[18] Performance and Scalability in DSS environment with Oracle9i, An Oracle Technical White Paper, April 2001, http://otn.oracle.com/products/Oracle9ipdf/dss_enviroment_9i.pdf

[19] Performance and Scalability in DSS environment with Oracle9i, An Oracle Technical White Paper, April 2001, http://otn.oracle.com/products/Oracle9i/pdf/dss_enviroment_9i.pdf

Pipelined Table Functions

Table functions are functions written in PL/SQL, Java, or C, which can accept cursors or sets of rows as input arguments and produce a set of rows as output.

Oracle allows table functions to pipeline results by returning result rows or subsets of rows incrementally for further SQL processing as soon as they are created, instead of waiting for the whole set to be completely constructed before being returned.

Oracle also provides support for parallel execution of table functions, allowing a set of input rows to be partitioned among multiple instances of a parallel function.

Pipelining and parallel execution of table functions help to improve the performance and scalability of a number of applications. In a data warehouse environment, table functions can be used in the ETL (Extraction-Transformation- Load) process that builds a data warehouse by extracting data from an OLTP system. The extracted data passes through a sequence of transformations before it is loaded into a data warehouse. In Oracle, the transformations can be pipelined and executed in parallel using table functions, avoiding the cost of materializing the outputs of each transformation in intermediate staging tables. The whole process becomes more efficient, more scalable and non interruptive.

SQL Server 2000 does not provide support for pipelined table functions.

CONCLUSION

According to the IDC report, "Worldwide RDBMS 2004 Vendor Shares: Preliminary Results for the Top 5 Vendors Show a Solid Boost (March 2005, IDC #32969), "*Oracle continues to be the overall leader in the worldwide relational and object-relational database management systems software market*". This supremacy is explained by features designed to meet and exceed the requirements of the most demanding environments, including e-business, OLTP, and data warehousing applications.

Oracle's unique features build on years of technical innovation and further extend the Oracle leadership by allowing all types of applications to perform and scale to exceptional standards.

Technical comparison of Oracle vs. SQL Server 2000: Focus on performance
May 2005
Author: Hervé Lejeune
Contributing reviewers: Vineet Buch, Sandra Cheevers, Rick Greenwald

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 © 2002 Oracle Corporation
All rights reserved.