Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance

An Oracle White Paper
February 2002

ORACLE

Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance

Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance

EXECUTIVE OVERVIEW

Recent years have seen entire corporations and organizations being integrated and streamlined, bringing everyone online -employees, business partners, customers, and suppliers- to access and share the latest information. These dramatic changes bring new challenges to databases: they must be able to service unprecedented numbers of users, and at the same time ensure good and predictable performance for a wide variety of concurrent applications, from OLTP applications with high transaction volumes to large Data Warehousing applications with complex queries.

This paper reviews the most significant differences between the market-leading database management system, Oracle9*i* Database, and its competitive product, IBM DB2 UDB V7.2, in the arena of performance and scalability.

INTRODUCTION

Oracle9*i* is Oracle9*i*. What is IBM DB2?

Even though product availability is not directly linked to performance, true portability across a large variety of hardware and operating systems enables users to seamlessly upgrade or change their hardware systems without having to worry about changing, redesigning or rebuilding their applications. In other words, portability helps preserve the initial investments in application software and helps deliver performance consistency across multiple platforms.

Because Oracle9*i* Database is one single product, with a single code base ported on a large selection of hardware and operating systems, it provides the same support for performance features and tools across all platforms. But while IBM promotes DB2 as one product, it is in fact a family of products, with different code bases, that differ significantly from each other in terms of features and tools.

For clarity and simplicity, this paper only considers DB2 UDB V7.2 EEE[1]. Throughout this document, the term DB2 will refer to the DB2 UDB V7.2 EEE

[1] DB2 UDB EEE is a superset of DB2 UDB EE: all features supported by DB2 UDB EE are supported by DB2 UDB EEE; some of the DB2 UDB EEE features described in this paper, e.g., partitioning, are not

product and the terms Oracle and Oracle database will refer to the Oracle9*i* Database product.

Key performance differentiators

The table and section below briefly describe the main differentiators between Oracle and DB2 in the arena of performance and scalability. These features are explained in more detail in the remaining sections of the document.

Feature	Oracle9*i*	DB2
Concurrency Model	Multi-version read consistency Non-Escalating row-level locking	No Locks escalate
Clustered configurations	Transparent scalability with Real Application Clusters	Rigid data partitioning required with DB2 EEE
Indexing capabilities	Wide variety of indexing schemes	Only B-Tree and dynamic bitmap indexes
Partitioning options	Range, hash, list and composite partitioning	Only hash partitioning
Additional data warehousing capabilities	MERGE Multi-table INSERT	Not supported Not supported
Intelligent advisories	Index, Summary, Memory, MTTR	Index advisory only
Self-tuning capabilities	Self-tuning memory, free space, and I/O management	No equivalent capabilities

Table 1: key differentiators

Concurrency Model

Oracle fully supports mixed workloads environments characterized by simultaneous query and update activities. With Oracle9*i* Database, writers never block readers and readers never block writers. Non-blocking multi-version read consistency always provides users with consistent query results while never imposing a performance penalty on concurrent update activity.

DB2 lacks Oracle's powerful multi-version read consistency and forces users to choose between accuracy and concurrency. This means that DB2 users

---

supported by DB2 EE.

must either block writers in order to ensure read consistency or accept inaccurate results, i.e., dirty reads.

The basic architecture of Oracle is very efficient for managing large numbers of transactions. The technical feature that makes this possible is Oracle's patented non-escalating row-level locking. All transaction processing applications reap the benefits of Oracle's technology. As more users are connected to the application, an Oracle database can continue to deliver consistent performance despite the increasing volume of transactions.

This efficient concurrency model is one of the reasons why, according to database scalability experts Winter Corporation, all 10 of the 10 largest Unix transaction processing databases[2] run on Oracle. No other database even comes close.

Due to the finite amount of memory structures available to track locking information, DB2 requires row locks to escalate to table locks to minimize resource usage when activity increases, leading to unnecessary contention and decreased throughput.

Support for clustered configurations

Real Application Clusters is the Oracle9*i* Database option that supports hardware clusters. It has been designed to provide full application compatibility. All types of applications: custom OLTP, DSS, or packaged applications such as the Oracle eBusiness Suite, can run unmodified when deployed from single systems to clustered configurations: no redesign or code changes are required; neither is explicit application segmentation or data partitioning.

In contrast, existing DB2 databases running on single systems must be migrated to be used with DB2 UDB EEE; this migration requires rigid data partitioning, and expensive and complex additional development.

Indexing capabilities

Oracle supports a wide variety of indexing schemes tailored for different types of operations and applications. In particular, Oracle supports static bitmap indexes and bitmap join indexes, which benefit data warehousing applications by providing dramatic response time improvements and substantial reduction of space usage compared to other indexing techniques. Additionally, Oracle supports global indexes across partitions, which are essential when using partitioned tables in OLTP environments.

---

[2] http://wintercorp.com/VLDB/2001_VLDB_Survey/winners/table11.html

DB2 only supports b-tree indexes[3] and dynamic bitmap indexes. DB2 equi-partitions indexes and tables, and does not support global indexes across partitions.

Partitioning options

Oracle9*i* Database offers several partitioning methods designed for different situations: range partitioning, hash partitioning, list partitioning, and composite hash-range and range-list partitioning.

DB2 only provides support for hash partitioning.

Additional data warehousing capabilities

Oracle provides two additional features useful in data warehousing environments, in particular during the Extraction, Transformation and Loading (ETL) process. The multi-table INSERT statement allows users to insert rows into multiple tables as part of a single DML statement, avoiding the use of separate SQL statements for each table. The MERGE SQL statement provides the ability to update or insert rows conditionally into a table or a view, depending upon which is needed.

DB2 does not support multi-table inserts and does not support MERGE.

Intelligent advisories

Oracle offers a number of, intelligent advisories for performance tuning that free the administrators from time-consuming tuning and diagnostics tasks and allow them to simulate a variety of "what-if" scenarios: index advisory, summary advisory, memory advisory, MTTR advisory, table/index usage advisory.

DB2 only provides an index advisory.

Self-tuning capabilities

Oracle provides many self-tuning capabilities that dynamically adjust the database parameters to take advantage of variations in the consumption of system resources. These capabilities include self-tuning memory management, automatic free-space management, and self-tuning IO management.

DB2 does not provide equivalent self-tuning capabilities.

TRANSACTION PROCESSING

Online transaction processing (OLTP) applications are characterized by very large user populations concurrently accessing large volumes of data for short and frequent insert or update transactions.

---

[3] "DB2 is limited to the use of Btrees for conventional indexing", p. 84, Database Report, Bloor Research, 2001

Such environments require support for high throughput, a choice between several indexing strategies, and excellent data concurrency. The Oracle9*i* Database provides unique features that make it the platform of choice for addressing these requirements.

Concurrency Model

Oracle and DB2 greatly differ in their implementation of concurrency control. The main differences are summarized in the table below and further explained in the following sections.

Oracle9*i*	DB2
Multi-version read consistency	Not available
No read locks	Requires read locks to avoid dirty reads
No dirty reads	Dirty reads if not using read locks
Non-escalating row-level locking	Locks escalate
Readers don't block writers	Readers block writers
Writers don't block readers	Writers block readers
No deadlocks under load	Deadlocks may be experienced 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 twice a query 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. Rather than

locking information to prevent it from changing while being read, or to prevent queries from reading changed but uncommitted information, 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 consistent version of the information can always be returned to any user.

DB2 does not provide multi-version read consistency. Instead DB2 requires applications either to use read locks, with various levels of isolation, or to accept dirty reads. Read 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. The only alternative users have is to build separate workload environments. The result is that DB2 users always have to find some compromise in their application design in order to get acceptable data concurrency and accuracy. IBM admits to this fact in their own documentation: *"Because DB2 UDB requests an exclusive lock on behalf of the application during an update, no other applications can read the row (except when the UR [uncommitted read, meaning dirty read,] isolation level is used). This can reduce concurrency in the system if there are a lot of applications attempting to access the same data at the same time. To increase the concurrency of the system, commit your transactions often, including read-only transactions. If possible, reschedule the applications that compete for access to the same table. Also, use Uncommitted Read [meaning dirty read] transactions where read consistency is not an issue."*[4]

With 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.

For an example of how this affects application development, consider SAP. In order to avoid the disastrous effects read locks could have on concurrency, SAP has to compensate for DB2 dirty reads. This is done through additional code implemented in the database-dependent layer of the SAP interface. In the Oracle interface for SAP, nothing extra has to be done to ensure read consistency since the database server takes care of it.

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.

---

[4] *IBM DB2 Universal Database Porting Guide, Oracle to DB2 v7.1, version 3.0*, p. 46, DB2 Technical Enablement Services, IBM Toronto Lab, November 2000

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.

DB2 also supports row-level locking as the default concurrency model. However, because it was not the initial 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 memory structures called lock lists. As for any memory structures, these lock lists have limited size and thus impose a limitation on the maximum number of locks that can be supported by the database.

As more users access the application and transaction volume increases, DB2 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.

"If a lock escalation is performed, from row to table, the escalation process itself does not take much time; however, locking entire tables decreases concurrency, and overall database performance may decrease for subsequent accesses against the affected tables."[5]

With DB2, *"A more important side effect of lock escalation is the concurrency impact on other applications. For example, if an application holds a share table lock on table T1 due to a lock escalation…other applications will not be able to update rows in T1."* [6]

An article in DB2 Magazine states that *"with … ERPs, lock escalation is one of the biggest contributors to poor performance."*[7] The article goes on to advise turning off lock escalation. While this is an option for DB2 on OS/390, lock escalation cannot be turned off for DB2 on Unix/Windows.

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. Such unpleasant deadlock situations are usually solved by aborting one or more of the concurrent users' transactions.

"An application ported directly from Oracle to DB2 UDB may experience deadlocks that it did not have previously".
November 2000, IBM DB2 Porting Guide, page 47

"As a result of different concurrency controls in Oracle and DB2 UDB, an application ported directly from Oracle to DB2 UDB may experience deadlocks that it did not have previously. As DB2 UDB acquires a share lock for readers,

---

[5] P. 266, DB2 UDB V7.1 Performance Tuning Guide, IBM Redbooks

[6] P. 47, IBM DB2 Porting Guide, November 2000.

[7] DB2 Magazine http://www.db2mag.com/db_area/archives/1999/q2/99sp_yevich.shtml

updaters may be blocked where that was not the case using Oracle. A deadlock occurs when two or more applications are waiting for each other but neither can proceed because each has locks that are required by others. The only way to resolve a deadlock is to roll back one of the applications." [8]

Oracle never escalates locks and, as a consequence, Oracle users never experience deadlock situations due to lock escalation.

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 DB2 support traditional B-Tree indexing schemes, but Oracle provides many additional indexing capabilities, suitable for a wider variety of application scenarios:

Feature	Oracle	DB2
Reverse Key Indexes	Yes	-
Function-based Indexes	Yes	Partial
Dynamic Bitmap Indexes	Yes	Yes
Stored Compressed Bitmap Indexes	Yes	-
Bitmap Join Indexes	Yes	-
Index-organized Tables	Yes	-

Table 3: Indexing Capabilities

Oracle provides the ability to create reverse key indexes. A reverse key index, compared to a standard index, reverses the bytes of each column indexed while keeping the column order. Such an arrangement can help avoid performance degradation when modifications to the index are concentrated on a small set of leaf blocks. By reversing the keys of the index, the insertions become distributed across all leaf nodes in the index.

DB2 does not support reverse key indexes.

With Oracle, indexes can also be created on functions of one or more columns in the table being indexed. A function-based index pre-computes the value of the function or expression and stores it in the index. A function-based index can be created as either a B-tree or a bitmap index.

[8] P. 47, IBM DB2 Porting Guide, November 2000.

With DB2's generated column feature, an index can be created based on the expression used to derive the value of the generated column. However, this implementation is less efficient than Oracle's function-based index because DB2 requires storage of the derived values in the table.

Finally, Oracle supports static bitmap indexes and bitmap join indexes. DB2 only supports dynamic bitmap indexes. These indexing schemes are explained further in the data warehousing and decision support section.

Index-organized tables

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 primary key index. 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 ROWIDs, which store the addresses of rows in ordinary tables and are used in conventional indexes to link the index values and the row data. Index-organized tables support full-table functionality, including ROWID pseudo-column, LOBs, secondary indexes, range and hash partitioning, object support and parallel query. 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.

DB2 does not support Index-organized tables. With DB2 it is possible to have specified columns appended to the set of an index's key columns, which may improve the performance of some queries through index only access but does not provide the storage efficiency of Index-organized tables since columns are duplicated in both the table and the index.

Clustered systems

Clusters are groups of independent servers, or nodes, connected via a private network (called a cluster interconnect). The nodes work collaboratively as a single system. Clusters allow applications to scale beyond the limits imposed by single node systems. Both Oracle and DB2 provide support for clustered configurations but differ greatly in their architecture.

Oracle9*i* Real Application Clusters

Real Application Clusters (RAC) is the Oracle9*i* Database option that supports hardware clusters.

Oracle9*i* Real Application Clusters adopts a shared disk approach. In a pure 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.

Oracle9*i* 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. Oracle9*i* 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.

DB2 Shared nothing architecture

DB2 adopts the shared nothing approach. In pure shared nothing architectures, database files are partitioned among the instances running on the nodes of a multi-computer system. Each instance or node has affinity with a distinct subset of the data and all access to this data is performed exclusively by this "owning" instance. In other words, a pure shared nothing system uses a partitioned or restricted access scheme to divide the work among multiple processing nodes. This only works well in environments where the data ownership by nodes changes relatively infrequently. The typical reasons for changes in ownership are either database reorganizations or node failures.

Parallel execution in a shared nothing system is directly based on the data partitioning scheme. When the data is accurately partitioned, the system scales in near linear fashion.

On a superficial level, a pure shared nothing system is similar to a distributed database. A transaction executing on a given node must send messages to other nodes that own the data being accessed. It must also coordinate the work done on the other nodes to perform the required read/write activities. Such messaging is commonly known as "function shipping". However, shared nothing databases are fundamentally different from distributed databases in that they operate one physical database using one data dictionary.

Performance for OLTP applications

The difference in the architecture adopted in the two products has many consequences in terms of performance and scalability, summarized in the table below:

RAC	DB2 EEE
No two-phase commit required	Requires two-phase commit
Data cached in multiple nodes	IPC for every cross-partition access
Single probe for data	Multiple partition probes
Uniform load distribution	Load skew likely

Table 4: Clustering

Two-phase commit

Any transaction that modifies data in more than one partition on a DB2 system must use the two-phase commit protocol to insure the integrity of transactions across multiple machines. DB2 transactions have to write the prepare records at commit time, during the first phase of the two-phase commit, and can only proceed to the second phase when the first phase has completed. This increases the response time of the OLTP application.

In RAC, a commit only needs to wait for a log force on the node that is running the transaction. If that transaction had accessed data modified by other nodes in the cluster, these blocks are transferred using the high-speed interconnect without incurring disk I/Os. RAC does require a log force of modifications present in the block before transferring. However, even on an insert intensive benchmark such as the SAP Sales and Distribution Benchmark, only a very small percentage of these transfers are blocked by a log force (less than 5%). This is because the log of modifications to a block is continuously forced to disk in the background by the log writer well before it is needed by another node.

Data Caching

RAC uses the global cache service (GCS) to ensure cache coherency. The global cache service allows RAC to cache infrequently modified data in as many nodes that need the data and have space in their caches. Further access to this data can be performed at main-memory speeds.

DB2 systems, on the other hand, must use inter-process communication to access data from another partition even if it has not been modified since the last access.

Partition probes

DB2 equi-partitions the indexes and tables. This causes multiple partition probes for queries that do not result in partition pruning. For example, if the employee table is partitioned by employee number and there is an index on employee name, a lookup of an employee by name will require DB2 to probe the employee

name index in all partitions. The total work performed to lookup the employee by name grows with the number of partitions.

By contrast, RAC can execute the same query by accessing only the appropriate index pages in the single B-Tree employee name index.

Load Skew

DB2 systems may suffer from load skew for two reasons. First, the underlying data may not be evenly distributed in all partitions. This is especially true with low cardinality data. Second, the data accesses may be skewed to a small set of column values due to seasonal or daily trends even when the underlying data is evenly distributed.

RAC does not suffer from load skew because there is no single node that owns the data, all the nodes can access all the data.

Transaction Routing

It is possible to further improve performance on RAC by routing transactions to a subset of the nodes in a cluster. This improves data affinity and reduces inter-node communication. The routing can be performed easily through the use of service names in the Oracle Net configuration.

Routing transactions by function is more cumbersome with DB2 because it requires knowledge of the location of the data accessed by the transactions. It is also less flexible to changes in load because executing the transactions on more (or less) number of logical nodes without data redistribution will result in sub-optimal performance.

In some situations, a RAC system can also be configured using appropriate middleware to route requests based on the application's bind values. For example, a mail server may route email connections based on the user's login. For optimal effect, this requires that the underlying data also be partitioned based on the bind value using range or list partitioning. This is not possible to implement in DB2 because the user has no control over the placement of data (DB2 supports only hash partitioning; range and list partitioning are not supported).

Performance for packaged applications

Popular OLTP Applications such as those from Oracle, SAP, PeopleSoft or Siebel, have thousands of tables and unique global indexes.

These applications require global unique indexes on non-primary key columns for speedy data access as well as for ensuring data integrity. Without these indexes, mission-critical application data can be corrupted, duplicated or lost.

Applications also do not partition their data accesses perfectly – it is not feasible to find partitioning keys for application tables that yield a high proportion of

"local" data accesses, where the requirements of a query can be satisfied exclusively by the contents of a single partition of data. Non-local data accesses incur the unacceptable performance overhead of frequent distributed transactions. With non-collocated data access, "*the data used by the transaction requires inter-partition communication in order to move the data across partitions. Things are even worse if this is an update transaction involving 2 or more partitions, which would add the 2-phase commit processing overhead.*"[9]

Most significant queries in SAP, PeopleSoft or the Oracle eBusiness Suite join multiple tables, and different queries use different alternate keys in the join predicates. To deploy a PeopleSoft or SAP application to a shared nothing database like DB2 would be a Herculean undertaking.

In contrast, packaged applications need not be re-written to run and scale against the Oracle9*i* Real Application Clusters architecture. As for any other application developed for Oracle on a single system, they require no porting or particular tuning effort to run on Oracle 9*i* Real Application Clusters: "*RAC provides […] virtually unlimited scalability, […] it supports all customers' applications without modification. Applications can benefit from the availability and scalability features of Oracle9i RAC even if they have not been designed specifically for RAC.*"[10]

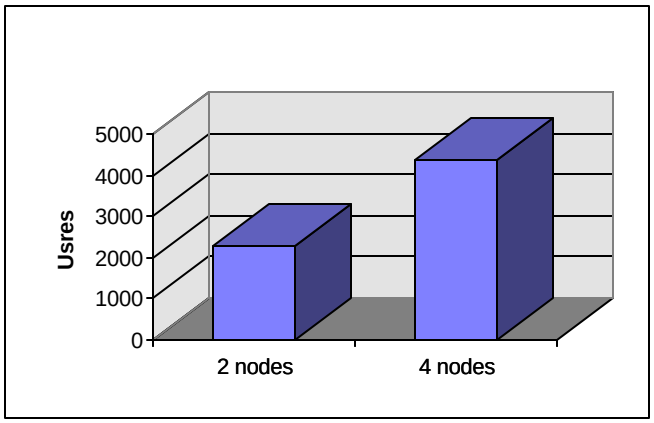

Figure 1: Oracle eBusiness Suite scalability on 9i RAC

This was clearly demonstrated using the Oracle Applications Standard Benchmark. Two benchmarks were run, respectively on two node and a four node cluster configurations, using the same base system. Results reported show an increase in the number of supported users from 2296 on the two node configuration to 4368 on the four node configuration, representing a scalability factor of 1.9.

---

[9] DB2 UDB EEE as an OLTP Database, Gene Kligerman, DB2 and Business Intelligence Technical Conference, Las Vegas, Nevada, October 16-20, 2000

[10] P.3, Implementing Oracle9*i* RAC with Linux on IBM @`server` xSeries servers, IBM Redpaper

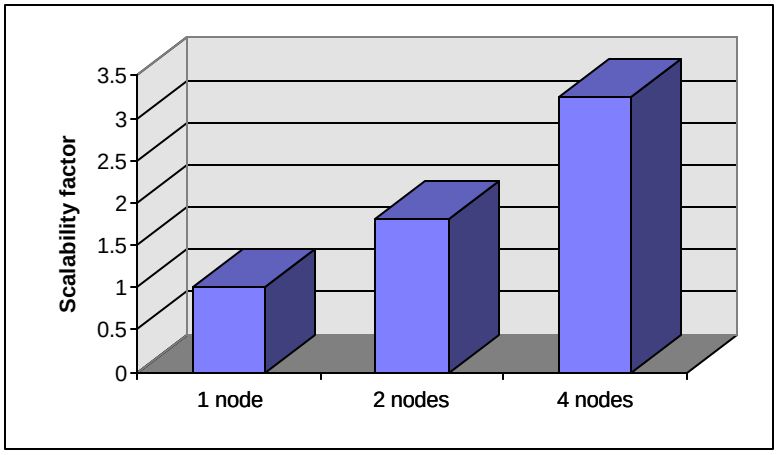

Figure 2: SAP scalability on 9i RAC

Similarly, initial performance tests conducted with the SAP R/3 Sales and Distribution (SD) application demonstrated an increase of performance by a factor of 1.8 when going from a one-node configuration to a two-node configuration, and by a factor of 1.8 when going from a two to a four-node configuration.

In both cases no particular application or database redesign was required.

According to the presentation, *'DB2 EEE as an OLTP Database'*, authored by IBM's Gene Kligerman and delivered at the International DB2 User's Group conference in Orlando, May, 2001, the performance of DB2 actually worsens as nodes are added in an OLTP environment. Kligerman says, *"When an environment is simulated where all transactions are uniformly distributed, the performance with 2 and 4 nodes is worse than with a single node."*

DATA WAREHOUSING AND DECISION SUPPORT

Data warehouses are very large databases specifically designed for query and data analysis. They 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, wide selection of partitioning strategies, and extended support for parallel execution. Here again, Oracle9*i* Database provides unique features that fully address these requirements.

Bitmap Indexes & Bitmap Join Indexes

Oracle supports static bitmap indexes and static bitmap join indexes.

A bitmap index uses a bitmap (or bit vector) for each key value instead of a list of ROWIDs. Each bit in the bitmap corresponds to a row in the table.

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.

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.

DB2 only supports dynamic bitmap indexes. Dynamic bitmap indexes are created at run time by taking the ROWID from existing regular indexes and creating a bitmap out of all the ROWIDs either by hashing or sorting.

For this reason, dynamic bitmap indexes do not provide the same query performance as Oracle's real bitmap indexes. While dynamic bitmap indexes can be used in "star transformation" strategies for executing star query, these indexes are still based upon b-tree indexes and there are considerable IO costs associated with accessing the much-larger b-tree indexes.[11]

Moreover, databases with dynamic bitmap indexes do not receive any of the space savings or index-creation time savings obtained by Oracle's true bitmap indexes.

Partitioning

Partitioning allows large database structures (tables, indexes, etc.) to be decomposed into smaller and more manageable pieces. Partitioning can help improve performance with the technique known as partition pruning. Partition pruning enables operations to be performed only on those partitions containing the data that is needed. 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.

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

[11] Key Data Warehousing Features in Oracle9*i*: A Comparative Performance Analysis, An Oracle White Paper, September 2001

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.

Finally, by enabling the parallel execution of DML statements, partitioning helps reduce response time for data-intensive operations on large databases typically associated with decision support systems and data warehouses.

Oracle's partitioning options

Oracle9*i* Database offers several partitioning methods designed to be more appropriate for various particular situations[12]:

- 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 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.

---

[12] For more information about Oracle9*i* 's partitioning options, see Oracle9*i* Partitioning, Hermann Baer, Technical White paper, Oracle Open World 2001 Berlin

DB2's partitioning options

The table below summarizes the differences between Oracle and DB2 with regard to the partitioning options that each product supports:

Feature	Oracle	DB2
Range partitioning	Yes	-
List partitioning	Yes	-
Hash partitioning	Yes	Yes
Composite partitioning	Yes	-
Local index	Yes	Yes
Global partitioned index	Yes	-
Global non-partitioned index	Yes	-

Table 5: Partitioning options

DB2 only supports the hash partitioning method, which has considerable limitations and weaknesses when compared to Oracle's partitioning capabilities.

Unlike range or list partitioning, hash partitioning does not allow typical queries to take advantage of partition pruning. By supporting more partitioning options for tables as well as indexes Oracle is able to prune partitions in more queries.

By only supporting hash partitioning, DB2 does not allow for 'rolling window' support. 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. DB2's hash partitioning scheme requires data in all partitions to be redistributed, therefore increasing the time required to load new data and also decreasing data availability as the table is locked during the data redistribution process.

Finally, DB2 requires equi-partitioning between tables and indexes, meaning that global indexes, partitioned or non-partitioned, cannot be created. This is a major problem in OLTP environments where global indexes are commonly used to offer efficient access to any individual record. With DB2, application designers have no flexibility when defining their indexing strategy in partitioned configurations.

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.[13]

DB2 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 performance and usability.

Multi-table inserts

Multi-table allows 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 extends 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[14] 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.

DB2 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.

---

[13] Performance and Scalability in DSS environment with Oracle9*i* , An Oracle Technical White Paper, April 2201

[14] Performance and Scalability in DSS environment with Oracle9*i* , An Oracle Technical White Paper, April 2201

PERFORMANCE TUNING AND TOOLS

Oracle and DB2 differ greatly in terms of diagnostics and tuning capabilities.

With Oracle9*i* Enterprise manager, users can quickly find which SQL statement is causing a performance problem, for example using the Top SQL capabilities. Then they can use SQL Analyze, the Index Tuning Wizard, and the Virtual Index Wizard to tune the statement and data access paths.

DB2 has no easy way of determining Problem SQL (using trace is the only way) and does not provide a tool to tune SQL by rewriting it

DB2, unlike Oracle, does not have a report that tells users which objects have statistics and when these statistics were generated. There is no easy way to find objects that need optimizer statistics.

In general, DB2 requires administrators to know a lot about the database. For example, to perform real time monitoring, DB2's Control Center provides administrators with a lot of metrics but without any precision about which ones are important indicators of the overall performance or health of the system. When confronted with a vague problem like "system is slow" the DB2 administrator has to know where to look and poke around to find the cause of the problem.

Oracle on the other hand guides the administrator via advice, help and drill-downs through a process of analyzing the cause of the problem. Oracle also provides more problem resolution capabilities, such as the advisors.

The following table and sections summarize the unique features provided by Oracle that enhance the information that can be used to tune databases, and help automate the tuning process. The absence of such features in DB2 requires administrators to use empirical approaches and manual interventions to tune the performance of the database.

	Oracle9*i*	DB2
Intelligent Advisories	Memory Advisors MTTR Advisor Summary Advisor Virtual Index Wizard	No equivalent features
Self-tuning capabilities	Self-tuning memory management Automatic free space management Self-tuning direct I/O management	No equivalent features

Table 6: advisories and self-tuning

Intelligent advisories

Memory

The System Global Area (SGA) is the group of shared memory structures that contain data and control information for an Oracle database instance. Program Global Areas (PGA) are private memory regions that contain data and control information specific to each server process. The sizes of these memory caches are configurable using initialization configuration parameters. Oracle provides a series of memory advisors that help administrators determine the optimal values for these configuration parameters.

The buffer cache size advisory mechanism allows administrators to size the buffer cache optimally by predicting the number of physical reads for different potential buffer cache sizes. The shared pool advisory mechanism displays information about estimated parse time savings for different sizes of the shared pool. The PGA target advisory helps administrators determine how key PGA statistics will be impacted if the overall amount of memory allocated to SQL working areas varies.

MTTR Advisory

Mean Time to Recover (MTTR) defines the desired amount of time required for Oracle to perform instance or media recovery on the database. The MTTR advisory helps administrators choose the optimal value by predicting the number of physical I/Os for different simulated MTTR values.

Summary Advisor

Oracle's 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 a materialized view, recommend materialized views based on collected workload information and report actual utilization of materialized views based on collected workload.

Virtual Index Wizard

The Virtual Index Wizard allows users to test and understand how a new index will affect SQL performance. Users can define an index and then without actually creating the index, understand how this index would affect the execution plan for an individual SQL statement.

Self-tuning memory management

SGA and Buffer cache

Dynamic SGA allows Oracle to set, at run time, limits on how much virtual memory Oracle will use for the SGA. Oracle can change its SGA configuration while the instance is running and both the buffer cache and the SGA pools can grow and shrink at runtime according to an internal, Oracle-managed policy.

Automated SQL execution memory management

Oracle provides an automated mechanism for dynamically allocating runtime memory to each query. Runtime memory is the memory region which is allocated during query execution for purposes such as sorting and hashing. In many data-warehouse environments, 70% or more of the data warehouse server's physical memory may be allocated for runtime memory.

This feature allows database administrators to specify the policy for sizing work areas. In automatic mode, working areas used by memory-intensive operators can be automatically and dynamically adjusted to compensate for low or high memory usage.

Automated SQL execution memory management offers several performance and scalability benefits for decision support workloads or mixed workloads with complex queries, that is, queries where a large portion of the runtime area is dedicated to working areas used by some memory intensive row sources such as sorts or hash-joins. The overall system performance is maximized and the available memory is allocated more efficiently among queries to optimize both throughput and response time. In particular, gains from improved use of memory translate to better throughput at high load.

Automatic Segment Space Management

With Oracle, it is possible to choose how free space can be managed inside database segments. With the automatic mode, the segment free/used space is tracked using bitmaps. A bitmap, in this case, is a map that describes the status of each data block within a segment with respect to the amount of space in the block available for inserting rows. As more or less space becomes available in a data block, its new state is reflected in the bitmap. Bitmaps allow Oracle to manage free space more automatically, and thus, this form of space management is called automatic segment-space management.

Bitmaps provide a simple and efficient way of managing segment space, optimizing space utilization, and providing better run-time adjustment to variations in concurrent access, and better multi-instance behavior in terms of performance/space utilization.

Self-tuning direct I/O management

Direct I/O involves transferring data blocks to and from the user process private memory directly, without involving the database's buffer cache. This feature improves performance by avoiding unnecessary pre-fetches, and by reserving I/O for the processes that can use them. Direct read uses a read ahead scheme to pre-fetch extents so that when the data layer clients need to access a requested block from disk, it would have been already pre-fetched.

The self-tuning direct IO feature in Oracle enables the server to dynamically adjust the number of direct reads based on the nature of the query, ensuring the optimal use of available bandwidth.

CONCLUSION

Oracle has a long history of bringing to market the best performing and most scalable database products.

Its latest release, Oracle9*i* Database, builds on years of technical innovation and further extends the Oracle leadership by providing new features and improvements that allow all types of applications to perform and scale to exceptional standards.

Technical comparison of Oracle vs. IBM DB2 UDB: Focus on performance
February 2002
Author: Hervé Lejeune
Contributing Authors: Jenny Tsai, Valerie Kane, Vineet Buch, Bill Kehoe, Sashikanth Chandrasekaran, Ray Glasstone

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.