Oracle Database Compression: A Competitive Overview

An Oracle White Paper
February 2008

ORACLE

Oracle Database Compression: A Competitive Overview

Oracle Advanced Compression: A Competitive Overview

INTRODUCTION

Exponential increase in data volumes in recent times has put enterprise IT infrastructures under severe pressure –from cost, performance, scalability as well as manageability perspective. It's become imperative to employ more efficient ways of storing and managing data to meet the growing demands being placed on IT systems. Database compression has been gaining momentum in recent years as a technique to reduce storage needs.

Oracle has been one of the pioneers in the field of database compression. Oracle Database 9i introduced Table Compression several years ago. Oracle Database 11g packages several new capabilities in its Advanced Compression option, providing a comprehensive set of functionality to optimize usage of storage, memory, and network bandwidth.

This paper presents a technical comparison between compression capabilities of Oracle Database and other vendors in the market.

KEY METRICS – WHAT REALLY MATTERS

Total storage cost and performance are two most important considerations for database compression. Compression ratio used in isolation may be misleading.

Before doing a comparison of various database features, it's important to first understand which metrics should be used to assess the costs and benefits associated with compression. The fundamental objective of using database compression is to minimize the total storage costs, while maintaining or improving the system performance. The key measures to assess the impact of compression in achieving this objective are described below:

Total Storage Cost

The most obvious metric to evaluate the benefit of a compression feature is the resulting reduction in storage cost. This should not only take into account the main production database but also all its copies such as such as standby, test, development and backup.

Compression Ratio

Compression ratio is often used to compare data size before and after compression. For example, if a 2 GB table were reduced to 0.5 GB by compression, the

compression ratio would be: 2 GB/0.5 GB = 4. Compression ratio is a good measure to provide a relative comparison of database size before and after compression. However, it can be often misleading to compare storage efficiency of different database products solely based on their compression ratios. This is due to the fact that databases have different internal structures for storing data and the original size for the same data can vary significantly across different products. It's the absolute size of the compressed database, and not the ratio alone, that ultimately determines the cost savings and other associated benefits. Consider the following illustrative example:

Database	Original Size (GB) (for same data)	Compression Ratio	Final Size (GB)
X	150	3	50
Y	100	2.5	40

In this example, database Y provides a lower final size even though database X appears to have a better compression ratio.

Performance

Best compression techniques provide performance benefits in several scenarios.

Impact on system performance should be carefully analyzed when considering database compression. While storage reduction provides cost savings, performance compromises are often unacceptable from a user experience perspective. The best compression methodologies not only provide space savings, they also maintain or improve performance. Some of the key parameters to assess performance impact are:

CPU overhead for compression

This refers to extra CPU cycles required for compression of data. Good compression techniques minimize this overhead.

CPU overhead for decompression and read performance

Some compression techniques may involve significant CPU cycles when compressed data is decompressed for reading. On the other hand, advanced techniques allow data to be read in compressed format and involve little or no 'decompression overhead'. They may, in fact, improve read performance as disk I/Os are reduced by virtue of more data being available per block.

DML performance

Apart from CPU cycles, the elapsed time required for DML operations such as INSERT and UPDATE is another defining characteristic of a compression algorithm. Addition of new data and modifications of existing compressed data should be intelligently managed by the database such that end users see minimal performance impact.

ORACLE DATABASE AND IBM DB2

This section compares compression features of Oracle Database and IBM DB2 for Linux, Unix and Windows (DB2 LUW).

The Starting Point - Data Storage Efficiency Before Compression

Oracle Database's native storage structure includes several space savings techniques that make the database size smaller to begin with, when compared with DB2.

Even before compression, there are significant differences in the way data is internally stored in Oracle Database and DB2. Oracle's storage structure focuses on efficiency and utilizes only the minimum required space for each data type.

For example, numeric data is stored with variable length in Oracle Database (NUMBER data type) while DB2 stores it in a fixed length format (DECIMAL data type). A DECIMAL(16,2) data type will be stored using a fixed length field required for 16 digits by DB2, no matter what the actual number of digits is[1]. The corresponding data type for Oracle Database is NUMBER (16,2). Oracle Database uses only the minimum required space to store NUMBER data type, depending on the actual length of data being stored.

For example, number 1.23 will use much less space than number 1234567.89, when stored in Oracle Database as NUMBER(16,2). For DB2, both of them will take the full space required for 16 digits. For many applications, such data can be a significant proportion of the total data stored, sometimes as high as 90%. Hence, this can lead to DB2 using significantly more disk space than Oracle Database for equivalent datasets.

Similarly, while storing variable length character data using VARCHAR, there is more overhead with DB2 than with Oracle Database. DB2 uses four additional bytes to store information about VARCHAR data[2], irrespective of the size of actual data. For example, a VARCHAR(10) field containing the string 'DB' will require a total of 2+4=6 bytes of storage in DB2. On the other hand, Oracle Database's overhead for storing such data can be as small as one byte, depending on the actual data size. In this example, Oracle Database will use only one additional byte to store information and will require 2+1=3 bytes of storage.

To understand the impact of internal storage structures on real world databases, consider a table ORDERS containing one million rows with two columns, CUSTNAME and AMOUNT. One can manually calculate the size of two columns as shown in the table below. Note that this is only the size of data in two columns, as table and row level overheads have not been included for simplification purposes. The actual table size will be slightly higher in both cases.

---

1 https://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.dbobj.doc/doc/t0020098.html

2 https://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com.ibm.db2.luw.admin.dbobj.doc/doc/t0020098.html

Column Name	Data Type	Average Actual Length (across all rows)	Oracle Column Size[1]	DB2 Column Size[2]
CUSTNAME	VARCHAR(30)	10 characters	10.5 MB	13.4 MB
AMOUNT	DECIMAL(12,2)-DB2 NUMBER(12,2)-Oracle Database	5 digits	4.8 MB	6.7 MB

In the above example, DB2 uses 31% more space than Oracle Database to store the same data. Depending on the table structure, this difference can easily get pronounced to a point where DB2 uses 40-50% more space than Oracle Database to store an equivalent dataset in a more real world setting.

Compression Algorithm

Oracle's block level compression provides better performance and dynamic compression capabilities for lasting benefits, when compared with DB2's table level compression. It practically has no limitations on the number of values that can be compressed, while DB2's dictionary has a restricted length.

Most compression algorithms are based on creating 'symbols' that replace repetitive occurrences of a particular data pattern. These symbols are stored in a dictionary and used to interpret compressed data. The key difference between Oracle Database and DB2 compression is the level at which the symbol table (or compression dictionary, in DB2 terminology) is managed. DB2 stores a dictionary per table object while Oracle Database has a block level symbol table included in the block header.

Oracle has consciously chosen to implement the dictionary at the block level for several reasons:

Better Performance

A block level dictionary can provide significantly better performance as compared to a table level dictionary. As compression symbols are stored within the same block, reading compressed data does not require fetching any additional blocks. In case of DB2, operations on compressed data will need to refer to the table level dictionary, creating additional overhead.

Furthermore, 'local' nature of Oracle's compression provides greater fault tolerance as the effects of any block level problems are localized to just that block. Consider the same situation with a table level dictionary where the entire table may be inaccessible if the dictionary cannot be read for any reason.

---

[1] Bytes required by Oracle Database for decimal data of precision p can be calculated as round((actual length(p)+s)/2))+1, where s is 0 for positive numbers and 1 for negative numbers. An additional byte is required to store length.
CUSTNAME in Oracle Database: (10+1)*1000000/(1024*1024)=10.5 MB
AMOUNT in Oracle Database: (round(5)/2+1+1)*1000000/(1024*1024)=4.8 MB

[2] Bytes required by DB2 for a decimal data type can be calculated as truncate(precision/2)+1.
CUSTNAME in DB2: (10+4)*1000000/(1024*1024)=13.4 MB
AMOUNT in DB2: (truncate(12/2)+1)* 1000000/(1024*1024)=6.7 MB

Adaptive Compression

DB2's table level dictionary is a static read-only dictionary, which does not adapt with time. DB2's dictionary gets created the first time compression is turned on for a table. A static dictionary may lead to significant deterioration in compression ratio over time, if the new data arriving is different from the data used for creating the original dictionary. To update the dictionary, a table REORG is required, which may result in database downtime.

This problem is particularly acute in data warehouse environments where the contents of date fields change over time. Newly loaded rows will have new dates, which will not be present in the dictionary. The older dates, which are present in the dictionary, are purged out of the system over time.

For example, consider a data warehousing fact table storing sales information for the most recent two years. Obviously the sales date of the transactions will change every day, as does the shipping or return date of items. If the DB2's table level symbol table was generated in December 2005, it potentially covers all or most date values for calendar year 2004 and 2005 (the two most recent years). However, it will not reflect any date values starting with 2006. By at the end of 2007 (or even earlier), there will be no usable compression values for the date information anymore. This will result in reduced compression ratio.

Consider another example of an electronics good retailer. Two years ago, this retailer compressed the sales table in their database with DB2's static compression scheme. However, for the past holiday season, the retailer's most-popular selling product was Apple iPod, a product which only recently emerged as a bestseller. Since iPod was not a commonly sold item when the sales table was originally compressed, the iPod product code is not part of the static symbol table, and thus none of this new sales data is being compressed. So, while the sales table was originally able to be compressed by a ratio of 2:1, new sales trends (such as the emergence of the iPod) has reduced the compression of new records to 1.2:1.

Similarly, a new customer of a telecom company would probably call a few close friends many times a month. This could generate a lot of redundancy in the phone bill, which can be eliminated through compression. DB2's old and static dictionary may not carry these numbers and hence such data may not get compressed.

Oracle, on the other hand, dynamically adapts to changes in data distribution thereby providing consistent compression. As new data is added and more space is allocated to the table, new block level symbol tables are created to compress newly added data. Further, any changes in the existing data via UPDATEs or INSERTs are also incorporated into its symbol table in a dynamic fashion – a process completely transparent to the administrator and end users. Oracle even reorders the column storage within a data block to enable transparent multi-column compression, thus adaptively optimizing the compression.

No Minimum Data Requirements for Initiating Compression

Another fallout of the static table level dictionary is that DB2 cannot create an effective dictionary until is has sufficient representative data from which to derive symbols at the table level. At the time compression is turned on, if data in the table is too small in size, the compression dictionary will either not be created or be ineffective as future data may be quite different. The administrator needs to wait till the table is sufficiently populated to create an effective dictionary. The Automatic Dictionary Creation feature of DB2 V9.5 can create a dictionary with 1-2 MB of data but this is may not be effective for data added in future. By the time this table reaches a reasonably large size, say 5 GB, the compression ratio may get significantly reduced, as the compression dictionary would still be based on the original 1-2 MB of data.

On the other hand, Oracle Database creates symbols for each block separately and hence can automatically create symbols relevant to the new data being added. One does not need to wait for the table to grow to a certain size to get long-term compression benefits.

No Additional Memory Required

DB2 needs compression dictionary to be stored in its database heap, which may require an increase in database heap size. Database heap in DB2 contains log buffer and catalog cache.

Further, at the time of dictionary creation, DB2 requires additional memory that comes from the utility heap. This may also entail increasing utility heap size. Utility heap in DB2 contains backup buffer and restore buffer, similar to large pool in Oracle. These requirements have been identified in SAP note 980067 as discussed later in this paper.

Oracle Databases compression does not require allocation of any additional memory as the dictionary is stored as a part of the block header.

Unlimited Dictionary Length

DB2 uses 12 bit symbols in its table level compression dictionary, which can provide only 4096 (2^{12}) unique symbols for the entire table. This can be very restrictive for large tables. If a large table has patterns that cannot be accommodated in this dictionary, the corresponding data will not get compressed. For example, an order table with 20 columns and a million records will have 20 million data values. DB2's dictionary may be insufficient to compress data across these 20 million data values as number of repeating values is likely to be much higher than 4096. This can leave significant amount of repeating data, which would be a good candidate for compression, uncompressed and result in low compression ratios.

Oracle Database's block level symbol table does not have any such restrictions. All blocks used for a table have their own symbols and collectively provide a practically unlimited symbol table for the table data.

Index Compression

Index Key Compression

Indexes can consume a large share of database storage and Oracle's index key compression can help reduce that significantly. DB2 does not offer index key compression. Also, Oracle's bitmap indexes provide large space savings when compared to DB2, which does not offer true bitmap indexes.

Index key compression allows compression of key column values in an index or index-organized table, thereby reducing the storage for repeated values. Oracle Database has the capability to compresses index keys while DB2 doesn't.

Generally, index keys are comprised of two parts – a grouping piece and a unique piece. If the key definition does not include a unique piece – as is the case with non-unique indexes, Oracle Database creates one by appending the rowid to the grouping piece.

Key compression splits the index key into a prefix (grouping piece) and a suffix (unique piece). If prefix values are repeated, they are stored only once and suffix entries share this prefix value. Each suffix entry references a prefix entry stored in the same index block.

By default, the prefix consists of all key columns excluding the last one. For example, in a key made up of three columns (column1, column2, column3) the default prefix is (column1, column2). For a list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4) the repeated occurrences of (1,2), (1,3) in the prefix are compressed.

Alternatively, one can specify the prefix length, which is the number of columns in the prefix. For example, if prefix length 1 is specified, then the prefix is column1 and the suffix is (column2, column3). For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4) the repeated occurrences of 1 in the prefix are compressed.

Index compression can result in significant storage savings and may also lead to performance improvements. If designed carefully, a compressed index uses a smaller number of leaf blocks resulting in less I/O and thereby improving performance.

SAP note 1109743 (Use of Index Key Compression for Oracle Databases) available at http://service.sap.com/notes[1] documents the impact of using Oracle Database's index key compression:

- *"Saves disk space for indexes and reduces total database size on disk*

  Customer experiences show that up to 75% less disk space is needed for key compressed indexes. On average you can reduce your total database size by 30% by using key compression for the largest indexes.

  Real world example: The size of 'GLPCA~1' index was reduced from 18GB to 4.5GB.

[1] Access authorization required

- *Reduces physical disk I/O and logical buffer cache I/O improving buffer cache quality*
- *Higher CPU consumption*

  Every compression technique comes with higher CPU consumption. The higher CPU consumption is more than compensated by doing less logical I/O for index blocks in the database buffer cache.
- *Improved overall database throughput*

  Early customer experiences have shown a 10-20% better database throughput for an SAP system by using index key compression in a non CPU bound environment."

Bitmap Indexes

Bitmap indexes are compressed by definition and do not require compression to be turned on separately. A bitmap index uses a bitmap (or bit vector) for each key value instead of a list of row identifiers. Each bit in the bitmap corresponds to a row in the table.

For example, consider an employee table as follows:

Employee Id	Name	Gender
1	John	Male
2	Martha	Female
3	Carol	Female
4	Bob	Male

A bitmap index on Gender column will conceptually appear as below:

	Bitmaps	
Employee Id	Male	Female
1	1	0
2	0	1
3	0	1
4	1	0

Here, the bitmap for value 'Male' (1,0,0,1) signifies that value of Gender column is 'Male' for rows 1 and 4.

As the bitmap representation is very efficient from a storage perspective, bitmap indexes can save a lot of space over other index types, especially for low and medium cardinality data and also provide significant performance improvements. For columns where each value is repeated hundreds or thousands of times, a bitmap index typically could be less than 25% of the size of a regular B-tree index.

Oracle supports persistent (a.k.a. static) bitmap indexes and persistent bitmap join indexes. 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. For each value in a column of a table, a bitmap join index stores the row id's of corresponding rows in one or more other tables.

Different from persistent bitmap indexes, 'dynamic bitmap indexes' are created by taking the row id from existing regular indexes and creating a bitmap either by hashing or sorting during query processing. For this reason, databases with only dynamic bitmap indexes do not receive any of the space savings obtained by Oracle's true bitmap indexes. DB2 supports only dynamic bitmap indexes while Oracle supports both persistent and dynamic bitmap indexes.

Compression of Unstructured Data

Oracle's SecureFiles compression and de-duplication reduces unstructured data size tremendously, a feature not available with DB2.

Over last few years, unstructured data such as documents, spreadsheets, images, video and audio files, etc. has been growing at a rapid pace. Web 2.0 and associated growth in collaborative Internet applications has been a significant driver of this trend. This growth is also driven by the regulations such as Sarbanes-Oxley, HIPAA, etc. that mandate storing of unstructured content to meet various legal requirements.

Traditionally, unstructured data has been stored using file systems. Relational databases were primarily used for storing structured data such as numbers and text. This dichotomy presents a significant challenge for applications which are increasingly being expected to combine the two types of data and present a seamless experience to end users. Architectures that pull unstructured data from file systems and mash it up with structured data from the database do not provide the level of security, scalability, transaction consistency and high availability for unstructured data that is available with Oracle Database.

Oracle Database 11g has introduced SecureFiles, which provides breakthrough performance that is comparable, and many a time even better, than file systems. SecureFiles presents a 'best of both worlds' scenario for storing and managing unstructured data. Using SecureFiles, applications can get the high level of performance earlier available only with file systems, and also attain the same level of security, scalability, transaction consistency and high availability as structured data within Oracle Database.

For today's applications, unstructured content can represent a significant proportion of data from a storage perspective, and can even exceed the structured data stored in the database. Besides providing breakthrough performance, SecureFiles also enables compression of unstructured data. Two levels of compression are available to enable customers make a trade off between compression CPU overhead and compression ratio. SecureFiles compression can result in huge storage savings for applications storing unstructured data inside the database. Additionally, with SecureFiles de-duplication functionality, multiple

copies of a document can be replaced by references to a single copy. This can provide a tremendous reduction in storage footprint for certain applications such as content management systems, email, etc. that typically have duplicate content. This can also improve performance of write and copy operations involving duplicate content as content only needs to be written once.

More details on Oracle SecureFiles are available at http://www.oracle.com/technology/products/database/securefiles/index.html.

DB2 does not offer compression of unstructured data. This can be a limitation for applications/environments that deal with significant volumes of unstructured data.

Network Traffic Compression

Oracle Database's network traffic compression allows standby databases to synchronize faster. DB2 does not offer this capability.

Oracle Data Guard provides the infrastructure to create, maintain, and monitor one or more standby databases. Data Guard maintains synchronization of primary and standby databases using redo data (the information required to recover a transaction). Network or standby server outages can prevent redo data from being transported to the standby server. When the outage is resolved, Oracle automatically performs redo gap resolution by transporting all redo data needed to synchronize the standby database.

Oracle Advanced Compression introduces the capability to compress redo data as it is sent over the network during redo gap resolution. Through this compression, network bandwidth efficiency is maximized to increase the gap resolution throughput. Gap resolution can be up to two times faster with compression – ensuring that the standby database is quickly synchronized and high availability is achieved.

DB2 does not provide such network compression capabilities.

Example – Impact on SAP Tables

SAP note 980067 indicates several performance issues with DB2's compression.

SAP note 980067 (DB6: Using DB2 9 Row Compression) available at http://service.sap.com/notes[1] identifies the disadvantages of using DB2 compression. While a few of these are generic manifestations of compression, many are caused by the specific design of DB2's table level compression technique and are not seen in Oracle Database compression. The table below compares the behavior in these areas:

DB2 Behavior	Oracle Database Behavior
"*Compressing and decompressing puts additional workload on the CPU. If your database server is bounded on CPU power, enabling deep compression could affect performance negatively.*"	The algorithm used by Oracle Database keeps this overhead very low which is often not noticeable to end users.

[1] Access authorization required

DB2 Behavior	Oracle Database Behavior
"Any operation on the database that accesses the table itself becomes slightly slower, because the rows need to be compressed or decompressed. The overhead for decompression might slightly slow down: • *SQL statements that perform table scans* • *RUNSTATS that reads the complete table and thus needs to decompress all rows to generate the statistics."*	Since Oracle compression is block based, it has little or no decompression overhead. Table scans are actually much faster on compressed data, due to reduced I/O. Time to gather table statistics is not impacted by compression in Oracle Database.
"REORG with RESETDICTIONARY requires an additional table scan to create the dictionary."	Oracle Database's block level compression does not require any manual updates to the dictionary, which is dynamically self-tuning.
"The dictionary for compressing and decompressing rows is stored in the database heap (DBHEAP). Therefore, you might need to increase the size of your DBHEAP by increasing the DBHEAP_SZ database configuration parameter."	No additional memory allocation is required.
"For the creation of the dictionary, DB2 requires 10 MB memory in the utility heap. Therefore, we recommend that you increase the utility heap by setting the database configuration parameter UTIL_HEAP_SZ to at least to 2500 pages."	No additional memory allocation required.
"The compressed rows can become very small. Therefore, you should enable the large RID feature of DB2 9 as well, so that all space on the data page can be used." [1] It should also be noted that large RID feature in DB2 may result in larger indexes, which can occupy additional space. Indexes on tables with large RID require two additional bytes per row entry[2].	When rows become smaller after compression, Oracle Database packs more rows in the same block without enabling any special feature. There is no limitation on number of rows that can be accommodated in the block, so long as there is free space.

[1] Versions prior to DB2 9 have a limitation of 255 rows per page. DB2 didn't fit more than 255 rows in a page even if space was available. For overcoming this limitation in DB2 9, large record identifier (RID) feature needs to be used. If a page has 255 rows and large RID feature is not used, compression for this page will yield no benefit for DB2 as no new rows can be added even if there is free space on the page.

[2] http://www.ibm.com/developerworks/db2/library/techarticle/dm-0702nesiem/

DB2 Behavior	Oracle Database Behavior
"Since compressed rows generally have different sizes, updates on the table can result in an increase of fragmentation and overflow pages as well as in the need for reorganization using REORG."	Oracle Database's compression algorithm ensures minimum fragmentation and no manual REORG is required.

ORACLE DATABASE AND MICROSOFT SQL SERVER

The latest available release of SQL Server (SQL Server 2005) does not have data compression capabilities and hence this paper does not attempt to compare it with Oracle Database. It may be noted that Microsoft has published plans of adding some compression capabilities in the next release (SQL Server 2008) but there is no formal document available at this time to enable a comparison.

ORACLE DATABASE AND TERADATA

Teradata provides a smaller and restrictive set of compression capabilities when compared with Oracle Database.

Teradata introduced its 'Multi-Value Compression' in Teradata Database V2R5.0. Teradata Database provides 'field level compression' wherein repetitions of a pattern within a column are compressed. It includes a bit field in every row header to indicate whether or not a field is compressed. There is a trade off between the storage required for this bit field and the savings from compression. This trade off should be carefully evaluated before compressing data.

This section compares compression capabilities of Teradata Database with Oracle Database.

Number of Values

Teradata Database can compress only 255 values per column[1], which can be very restrictive. For example, if an invoice table with 100,000 rows had 2000 distinct customer ids, Teradata Database will be able to compress only 255 of those. Oracle Database does not have any such restrictions.

Creation of Dictionary

Teradata Database requires that the compression dictionary be defined explicitly at the time of table creation. The DDL needs to contain values that are to be compressed for a particular column. This is an extra burden on the administrator/designer and also does not help compress new data that may be different from existing data. On the other hand, Oracle Database can automatically create the symbol table by analyzing block data and can maintain the symbol table in a dynamic fashion.

[1] http://www.teradata.com/t/page/86995/index.html

Data Type Support

Teradata Database offers compression only for a limited set of data types. It does not support the following data types: INTERVAL, TIME, TIMESTAMP, VARCHAR, VARBYTE, and VARGRAPHIC.

In contrast, Oracle Database has the capability to compress all data types. As discussed earlier, Oracle Database 11g also has the capability to compress SecureFiles used to store unstructured data such as documents, videos, audio files, images, etc. This capability is not available with Teradata Database.

Other Differences

Various other compression capabilities of Oracle Database, such as backup compression, network compression, index compression, etc. are not available with Teradata Database.

ORACLE DATABASE AND HARDWARE BASED COMPRESSION

Hardware compression is generic in nature and does not offer various benefits associated with Oracle Database's specialized compression capabilities.

Some of the storage vendors provide compression at a hardware level. Such solutions compress data right before it goes to the storage media and decompress it right after it is read off the media, making it transparent to applications. While such solutions may reduce storage requirements, this kind of compression is very generic in nature and has several disadvantages in the context of databases as discussed below:

Compression Granularity

The ability to pick data selectively for compression is restricted in hardware based compression. The finest possible granularity may be achieved at a file level at best. Using Oracle Database compression, data can be compressed for individual tables, table partitions, indexes, etc. depending on the specific needs.

Read Performance

Log structured file systems lend themselves the most to provide storage compression. But they have poor read performance as table data may be scattered in different parts of the disk. This can result in very poor performance for read intensive operations such as full table scans. As mentioned earlier, read performance with Oracle Database's compression may significantly improve in several cases, due to reduction in I/O.

Need for Decompression

Data compressed at the storage level will need to be decompressed before it's brought into memory. This will create additional CPU overhead and may result in poor performance. Oracle Database does not require data to be decompressed – it can keep data compressed in memory, thereby reducing I/Os and providing fast data reads.

Further, Oracle Database can pack more data into memory using compression, thereby improving memory efficiency. This benefit is not available using hardware compression as data gets decompressed before being read into memory. Better memory efficiency can also reduce RAC interconnect traffic - a benefit hardware compression cannot provide.

Similarly, updates on the data will be more expensive with hardware compression as decompression, update and recompression will be required to complete the operation. On the other hand, Oracle Database is aware of structure of the database block and can do compression of rows in batches, thereby significantly reducing any performance impact to update operations.

Cascading Benefits

With Oracle Database compression, data stays compressed in all environments. Hence, benefits of reduced storage requirements cascade down to all environments such as testing, backups, standby databases, disaster recovery sites, etc., irrespective of the hardware used. With hardware compression, the benefits will be selectively available only in those environments that use the compression enabled hardware.

CONCLUSION

Database vendors differ in design and evolution of their storage optimization strategies and Oracle has been a pioneer in the area of database compression. From the very beginning, Oracle Database has focused on efficient space utilization and balanced that carefully with performance. Oracle Database uses highly efficient native data storage techniques, even without compression. Using Oracle's compression technology, organizations can realize further benefits by compressing all types of data with no administrative overhead. These benefits cascade to all environments, providing manifold reduction in storage costs. Oracle Database undoubtedly provides a richer set of compression functionality than any other competing product, and not only reduces storage costs but also improves database performance.

ORACLE

Oracle Database Compression: A Competitive Overview
February 2008
Author: Ajeet Singh
Contributing Authors: Vineet Marwah, Sushil Kumar, William Hodak

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
oracle.com