Analysing Compression Performance for Real Time Database Systems

[1]M. Muthukumar & [2]T. Ravichandran

[1]Karpagam University, Coimbatore, Tamilnadu-641021, India,
[2]Hindusthan Institute of Technology, Coimbatore, Tamilnadu-641032, India

***Abstract* - Database compression and decompression is a susceptible problem in the database framework. Database compression is widely used in data management to improve the performance and save storage space. When you have large amounts of data, the cost of the storage subsystem can easily exceed the cost of your data server. Over the last decades, improvements in CPU speed have outpaced improvements in main memory and disk access rates by orders of magnitude, enabling the use of database compression techniques to improve the performance of database systems.**

In this paper, we outlined the new techniques to analyse the database compression performance for real time database systems. This proposed technique not only reduces space requirements on disk and I/O performance, they also reduce the utilization of memory, thus reducing the number of buffer faults resulting in I/O. This algorithm enables more granular, enhances the compression and decompression performance.

***Index Terms* – compression performance, database performance, database compression, database decompression, real-time database**

I. INTRODUCTION

My previous work describes how to optimize and enhance the database compression for real time database systems [1]. This work outlines an effective approach to analyse the compression performance for real time database systems. This can be achieved by exploiting efficient algorithm of ***Hierarchical Iterative Row-Attribute Compression (HIRAC).***The algorithm has different performance behaviour as a function of dataset parameters, sizes of outputs and main memory availability. The analysis and experimental results show that the algorithms have better performance than the traditional algorithms.

Computing database compression is a big challenge. Our goal is to develop effective and efficient algorithms to compress the large size databases. This paper presents the efficient algorithm for very large compressed database. The algorithm has different performance behaviour as a function of dataset parameters, sizes of outputs and main memory availability. The algorithm is described and analysed with respect to the I/O, memory and performance. A decision procedure to select the most efficient algorithm given and aggregation request is also given. The analysis and experimental results show the proposed HIRAC algorithm compare favourably with previous algorithms.

The very large scale databases normally have very large size and a high degree of scarcity. That has made database compression as very important. Research in this area has considered various aspects of the problem such as developing a model for database compression, decompression and maintaining them. Proposed approach presents a different challenge in database compression and decompression era.

A large number of compression schemes and techniques have been devised based on large database. The advantageous effects of data compression on I/O performance in database systems are rather obvious, i.e., its effects on disk space, bandwidth, and throughput. However, we believe that the benefits of compression in database systems can be observed and exploited beyond I/O performance. Database performance strongly depends on the amount of available memory, be it as I/O buffers or as work space for HIRAC algorithms. Therefore, it seems logical to try to use all available memory as effectively as possible in other words, to keep and manipulate data in memory in compressed form. In this report, we introduced techniques to demonstrate their effect on database performance.

The size of the compressed backup is smaller than that of the uncompressed backup, which results not only in space savings, but also in fewer overall I/O operations during backup and restore operations. The amount of space you save depends upon the data in the database, and a few other factors, such as whether the tables and indexes in the database are compressed, and whether the data in the database is encrypted.

The compression rates that can be achieved for any dataset depend, of course, on the attribute types and

value distributions. For example, it is difficult to compress binary floating point numbers, but relatively easy to compress English text by a factor. In the following, we do not require that all data is English text; we only require that some compression can be achieved. Since text attributes tend to be the largest fields in database files, we suspect that expecting an overall compression factor is realistic for many or even most database files. Optimal performance can only be obtained by judicious decisions which attributes to compress and which compression method to employ.

The rest of the chapters, we briefly indicate related work. In Section 3, we illustrate the proposed algorithms to manipulate database compression. Section 4 contains a various algorithms used for analysing the performance of areal time database systems. The experimental evaluations are presented in section 5 and performance results are presented in section 6. Conclusions are presented in section 7.

II. RELATED WORK

The compression of database systems for real time environment is developed with our proposed HIRAC algorithm. It provides good compression while allowing access even at attribute level [1]. Optimal multi storage parallel backup for real time database systems by using Parallel Multithreaded Pipeline (PMP) algorithm to store compressed database at multiple devices in parallel[2].Compression reduces both the number and size of records written to temporary files, resulting in a reduction of I/O costs on temporary files by a factor of six [3]. This technology trend has enabled the use of data compression techniques to improve performance by trading reduced storage space and I/O against additional CPU overhead for compression and decompression of data [4].

Order-preserving methods have considerable CPU overhead that offsets the performance gains of reduced I/O, making their use in databases infeasible [5].Significant part of the compression time is used for memory accesses. It is impossible to make a general statement how the time required for one memory access compares to the time for one computation [6].

A number of researchers have considered text compression schemes based on letter frequency, as pioneered by Huffman. Other recent research has considered schemes based on string matching [7] Comprehensive surveys of compression methods and schemes are given in. Others have focussed on fast implementation of algorithms, parallel algorithms and VLSI implementations [8].

Few researchers have investigated the effect of compression on database systems and their performance [9]. Other researchers have investigated compression and access schemes for scientific databases with very many constants and considered a very special operation on compressed data [10].

III. DATABAE COMPRESSION FOR REAL TIME DATABASE SYSTEMS

One of the big challenges in the world was the amount of data being stored, especially in Data Warehouses. Data stored in databases keep growing as a result of businesses requirements for more information. A big portion of the cost of keeping large amounts of data is in the cost of disk systems, and the resources utilized in managing the data. In this section, we outline how compression can be exploited in database systems, and how proposed algorithms used for optimizing the compression for real time data base systems.

3.1 HIRAC Algorithms

The compression of database systems for real time environment is developed with our proposed Hierarchical Iterative Row-Attribute Compression (HIRAC) algorithm, provides good compression, while allowing access even at attribute level. HIRAC algorithm repeatedly increases the compression ratio at each scan of the database systems. The quantity of compression can be computed based on the number of iterations on the rows. To compress database, an algorithm is presented here called HIRAC algorithm which iteratively enhances the collection of selected representative rows. From one step to the next, new representative rows may be chosen, and old ones discarded. Since the endeavour of a compression algorithm is to decrease the storage constraint for a database, Compression openly utilize this as an optimization principle and guarantee that only patterns which progress the compression are established in each step of its iterations. Though the optimization problem is hard in an existing compression algorithms case as well, the heuristic used in this simple. HIRAC algorithm approach provides much lower time complexity contrast to an existing technique.

The proposed HIRAC with parallel multi-storage backup for real time database systems comprises of three operations. The first operation is to identify and analyze the entire database, the next step is to compress the database systems to take up as backups and the last step is to store the compressed backups in multiple storages in parallel.

The first phase is to analyze the database based on the environment in which it creates. At forts, the attributes present in the database systems are analyzed and identify the goal of the database creation and maintain a set of attributes and tables maintained in the database systems.

The second phase describes the process of taking the backup of database system by compressing the database using Hierarchical Iterative Row-Attribute Compression (HIRAC). The HIRAC algorithm is used to provide a good compression technique by allowing access to the database level and enhances the compression ratio for a ease backup of database systems.

The third phase describes the process of storing the compressed backups at different levels of storages in parallel. Since it has been stored at multi-storage devices, the copy of compressed backups is always available at any system, there is less chance of database systems to be lost and can easily be recovered.

3.2 ILC Algorithm

Iterative Length Compression (ILC) algorithm provide the solution to how to optimize and enhance the process for compress the real time database and achieve better performance than conventional database systems. This algorithm provides a solution to compress the real time databases more effectively, reduce the storage requirements, costs and increase the speed of backup. The compression of database systems for real time environment is developed with our proposed ILC algorithm.

Data compression significantly increases CPU usage, and the additional CPU consumed by the compression process might adversely impact concurrent operations. On the plus side, backup sizes and backup/restore elapsed times can be greatly reduced. Database compression offers the following benefits.

- Reducing Storage Requirement
- Data Transfer Rate
- Enhancing Data Security
- Backup and Recovery
- Performance Enhancement

ILC algorithm provides the solutions for the above issues and repeatedly increases the compression ratio at each scan of the database systems. The quantity of compression can be computed based on the number of iterations on the rows. When introducing data compression technology into real-time database, two requests must be satisfied. First, the compression algorithm must provide high compression radio to realize large numbers of data storage in real-time database. Second, the compression algorithm must fast enough to satisfy the function of real-time record and query in real-time database.

IV. ANALYSING DATABASE COMPRESSION PERFORMANCE

Database compression can be exploited far beyond improved I/O performance. If the database contains compressed values, it is trivial to include compressed values in the log. In other words, compressing the database values also reduces the size of log records and therefore the I/O traffic to log devices. Thus, compressing database values improves I/O performance on both the primary database and the log. It is conceivable that the savings in logging alone justify the overhead of compressing database values to be entered into the database.

The goal of database compression is to represent the information with as few bits as possible. Therefore, each bit in the output of a good compression scheme has close to maximal information content, and bit columns seen over the entire file are unlikely to be skewed. Furthermore, bit columns will not be correlated. In a simple performance comparison, we have seen that for data sets larger than memory performance gains larger than the compression factor can be obtained because a larger fraction of the data can be retained in the workspace allocated to a query processing operator. Most obviously, compression can reduce the amount of disk space required for a given dataset. This has a number of ramifications on I/O performance.

First, the reduced data space fits into a smaller physical disk area therefore, the seek distances and seek times are reduced. Second, more data fit into each disk page, track and cylinder, allowing more intelligent clustering of related objects into physically near locations. Third, the unused disk space can be used for disk shadowing to increase reliability, availability, and I/O performance. Fourth, compressed data can be transferred faster to and from disk. In other words, data compression is an effective means to increase disk bandwidth, not only by increasing physical transfer rates but by increasing the information density of transferred data and to relieve the I/O bottleneck found in many high-performance database management systems. Fifth, in distributed database systems and in client-server situations, compressed data can be transferred faster across the network than uncompressed data. Uncompressed data require either more network time or a separate compression step. Finally, retaining data in compressed form in the I/O buffer allows more records to remain in the buffer, thus increasing the buffer hit rate and reducing the number of I/Os.

The last three points are actually more general. They apply to the entire storage hierarchy of tape, disk, controller caches, local and remote main memories, and CPU caches. If storage space on all levels is used more efficiently, bandwidth is saved when moving data up or down in the hierarchy, when moving data laterally between memories and caches, and by achieving a higher hit rate at each level. Reducing the amount of bus traffic in shared-memory systems might also allow higher degrees of parallelism without bus saturation.

The compression cost of each operator includes the CPU cost and the I/O cost. The CPU cost depends on the CPU speed, the compression method, and the input result size. The I/O cost includes the cost for reading the input and writing the output, which can be saved if the plan is pipelined in a unit that can be held in memory. For a serially executed plan, the compression cost equals the sum of the cost for each operator.

The decompression cost depends on the compression plan and the client's access pattern of the result. If the client decompresses the query result immediately, the decompression cost is just the cost to decompress the

whole result. It can be computed by adding up the cost for individual operators. If the client stores the result in a compressed form and needs N random accesses to certain units the decompression cost equals the cost to decompress.

V. EXPERIMENTAL EVALUATION

In this work we associated the compression performance for real time database systems using traditional and proposed algorithms. We used a real time 1GB sample database for a trialling to examine the efficiency of the system performance. We have outlined a number of performance effects of database compression in the next section in order to examine the efficiency of the proposed HIRAC algorithm.

System Component	Description
Processors	Intel® Core 2 Duo CPU P-IV 2 GHz Processer
Memory	3 GB memory
Operating System	Microsoft Windows XP Professional Version 2002 Service Pack 3
Database Manager	Microsoft SQL Server 2008 Enterprise Edition (64 bit)
Database Size	1 GB

Table 1: System Components

The proposed HIRAC based compression model for real time environment is efficiently designed for compression and taking backup compressed data with the database systems.

Most systems already employ "log compression" by saving only log records of committed transactions when archiving log devices on inexpensive storage media. In addition to these methods, we propose an inexpensive technique to reduce log traffic earlier in the process, namely before a log record is first written to disk. Compression not only improves I/O performance and hit rates at each "buffering" level in the storage hierarchy, it also can be exploited in database query processing without much change to implemented algorithms. Table 1 describes the details about system configuration used for analysing the compression performance.

VI. RESULTS AND DISCUSSION

The CPU cost for backup compression is significant and can impact concurrent operations unless some means is used to limit backup CPU usage. Database tables and indexes to be compressed generally should be chosen to reduce overall I/O rates and thus have a minimal impact on performance, unless spare CPU cycles are available. Database compression is effective in general, even with an already compressed database and reduces both storage and elapsed times for backup and restore.

Database compression affects the page on disk and in memory. It does not change the logical attributes of the data or the way it is presented by the database, so there are no changes visible to the application. Data compression requires more processing for select, insert, and update than for uncompressed data. Furthermore, compression is generally more expensive than decompression.

Compressing large database consume significant I/O volume, improve their memory caching and reduce the I/O volume enough to compensate for the compression/decompression overhead, thus reducing storage costs without undue change in performance. In certain I/O-bound situations, data compression can even improve overall performance.

6.1 Disk I/O Reduction and Performance Impact

When HIRAC compression is used, there is a multi-way trade-off involved between disk, buffer pool, I/O reduction and performance (loss due to compression/decompression overhead, but gain due to lower I/O rate). Generally analysing the database performance can be divided into three sections on I/O performance and buffering, transaction processing, and query processing. For transaction processing, there will probably be two main effects of compression. First, the buffer hit rate should increase since more records fit into the buffer space. Second, I/O to log devices should decrease since the log records can become shorter. The results below show performance of disk I/O ratios and storage utilization.

Compression Types	Disk I/O Reduction (%)	Performance Ratios
Uncompressed	100	1.00
RLE Compression	89	1.12
Dictionary Compression	84	1.19
ILC Compression	77	1.30
HIRAC Compression	71	1.41

Table 2: Disk I/O Reduction and

Performance Ratios

The above table (table 2) described the disk I/O reduction and performance ratio based on real time

sample database. The performance efficiency of compression using the proposed HIRAC algorithm is compared with existing compression algorithms. The following graph shows bigger I/O reduction and large improvements with proposed algorithm.

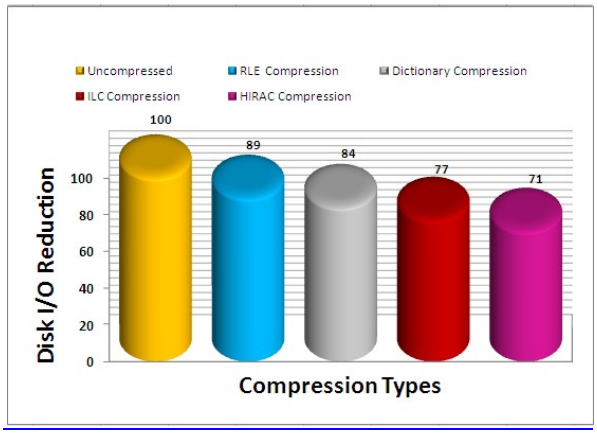

Fig. 1: Compression Types and Space Disk I/O Reduction

It is very encouraging to observe in this graph that very moderate compression factors reduce the total I/O cost significantly. Even if some additional cost is incurred for decompressing output data, the performance gain through compressed permanent and temporary data on disk and in memory far outweighs the costs of decompression.

Performance improved due to lower I/O rates and improvement in memory utilization. Note that the effect on performance was much more dramatic with less memory. Fig. 2 describes the process of performance based on different types of existing and proposed compression algorithms. Compared to an existing compression algorithm the proposed HIRAC algorithm achieves good performance ratios and the variance would be about 40% better.

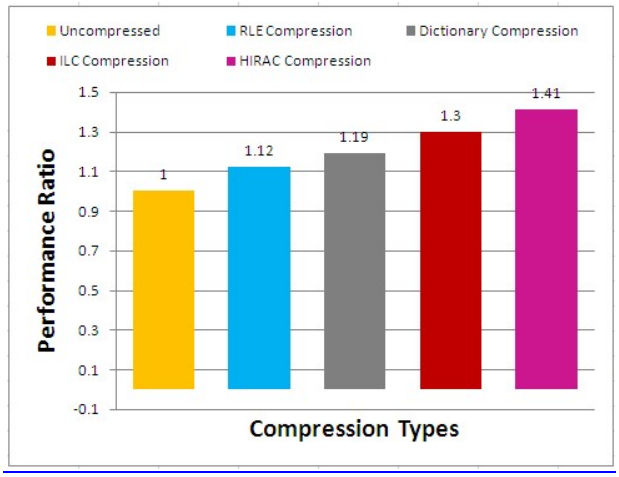

Fig. 2: Compression Types and Performance Ratio

The above graph shows performance improvement and these results demonstrate that at least some I/O bound situations have the potential for dramatic gains through use of data compression.

6.2 Memory Usage

The memory usage required to make compressed backups can be significantly less because the size of compressed backups are smaller and there are fewer writes to the backup media. If memory is very limited in a system, it might be more important to allocate it to operators that may be able to run without overflow, and to use memory there with maximal efficiency.

Compression Types	Average Memory Usage
Uncompressed	20762 K
RLE Compression	17648 K
Dictionary Compression	16559 K
ILC Compression	13756 K
HIRAC Compression	12458 K

Table 3: Compression and Average Memory Usage

The graph below shows that in situations where the system memory is insufficient for a large enough buffer pool, resulting in an I/O bound configuration that cannot keep the processors completely busy, being able to HIRAC compression the memory performance can improve dramatically.

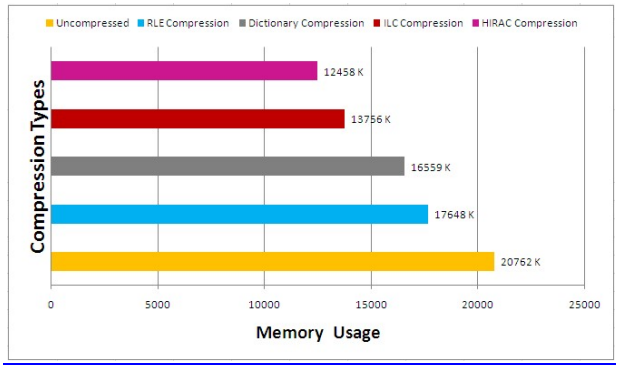

Fig.3: Compression Types and Memory Usage

Figure 3 shows the memory usage for large database. As in the above figures, even a small amount of compression improves the performance significantly. However, for small memory sizes compared to the build input, the effect of fitting more compressed records than uncompressed records in memory is not as strong as for medium-size relations.

VII. CONCLUSION

In conclusion, direct manipulation of database compression is an important for managing very large data warehouses. The key ideas are to compress attributes individually, to employ the same compression

scheme for all attributes of a domain and to perform data manipulations before compressing the data. Proposed approaches provides significant gains in cost model that takes memory performance, I/O benefits of compression and the CPU overhead of decompression into account.

The analysis and experimental results show that the proposed HIRAC algorithms have better performance than the traditional algorithms. Furthermore, implementing our techniques for database compression is also very easy.

VIII. REFERENCES

[1] M.Muthukumar and T.Ravichandran, Optimizing and enhancing parallel multi storage backup compression for real-time database systems, IJECTA, ISSN: 2229-6093, Volume3 Issues4, Pages 1406-1417, July 2012.

[2] M.Muthukumar and T.Ravichandran, Optimizing multi storage parallel backup for real time database systems, IJESAT, ISSN: 2250-3676: Volume2 Issues5, Pages 1515-1521, Sep 2012.

[3] G. Graefe and L. Shapiro.Data compression anddatabase performance.In ACM/IEEE-CS Symp. OnApplied Computing, pages 22–27, April 1999.

[4] Kesheng, W., J. Otoo and S. Arie, 2006. Optimizing bitmap indices with efficient compression, ACM Trans. Database Systems, 31: 1-38.

[5] ShivnathBabu, Minos N. Garofalakis, and Rajeev Rastogi. Spartan: Amodel-based semantic compression system for massive data tables. In*SIGMOD'2001*, 2001.

[6] Peter A. Boncz, Stefan Manegold, and Martin L. Kersten. Databasearchitecture optimized for the new bottleneck: Memoryaccess. In *Proc.of VLDB*, pages 54–65, 2004.

[7] KaushikChakrabarti, Minos N. Garofalakis, Rajeev Rastogi, andKyuseok Shim. Approximate query processing using wavelets. In*VLDB'2000*, pages 111–122, 2000.

[8] S. Amer-Yahia and T. Johnson.Optimizing queries oncompressed bitmaps. In *Proc. of VLDB*, pages329–338, 2000.

[9] G. Ray, J. R. Harista, and S. Seshadri. Databasecompression: A performance enhancement tool. In *the7th Int'l Conf. on Management of Data (COMAD)*,Pune, India, 1995.

10] Graefe, G., "Query Evaluation Techniques forLarge Databases", ACM Computing Surveys, 25(2),2003.