ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

Efficient Compression Techniques for an In Memory Database System

Hrishikesh Arun Deshpande

Member of Technical Staff, R&D, NetApp Inc., Bangalore, India

ABSTRACT: Enterprise resource planning applications require rapid response to queries, dynamic scaling and massive parallel processing something that traditional disk based databases are unable to satisfy. The advent of battery backed up non-volatile main memory and multicore parallel processing architectures together with increasing memory sizes and falling hardware costs has made it possible to realize a database that resides completely in the main memory. But the size of main memory being comparatively smaller than a hard disk, it's essential to compress the in memory data so as to achieve the desired space saving while at the same time leverage the rapid sequential access provided by main memory. This paper gives insights into the various compression techniques that can be used to encode an in memory database and the conditions under which a given technique can be advantageous over others.

KEYWORDS: in memory database; compression; attribute vector; dictionary; encoding; compression; scan performance; direct access.

I. INTRODUCTION

Enterprise Resource Planning (ERP) is being increasingly used to collect, store, manage and interpret business data in applications such as invoice generation, order tracking, sales forecasts among others. This requires massive parallel processing, scalability and faster response times something thattraditional databases are unable to guarantee [1]-[3]. Traditional databases are often optimized for disk storage and have higherseek times since queries require that data needs to be fetched all the way from the disk storage to the CPU's registers for processing. On the other hand, if all the required data is present in memory, then data access is faster resulting in rapid response to queries. Table 1 below demonstrates the average seek times when similar data packets are accessed by the CPU from different memory components [3].

Table 1. Data access seek time

Data Access	Seek Time
L1 cache reference	0.5ns
Branch Mispredict	5ns
L2 cache reference	7ns
Mutex lock/unlock	25ns
Main memory reference	100ns
SSD random read	150us
Read IMB sequentially from Memory	250us
Disk Seek	10ms

With the advent of massive parallel processing systems and a non-volatile Random Access Memory (RAM) it has become possible to realize an in-memory database (IMDB) where the entire data required for database operations resides in the system's main memory [3]. Such a database is also called a main-memory database (MMDB) [4]. As shown in Table 1 above, a data access from the main memory (100ns) is almost 100,000 times faster than a disk access (10ms). Accessing the data in memory greatly reduces the seek time during querying and provides a faster, predictable response in comparison to traditional disk oriented databases [5].Thus, a main memory resident database provides rapid data access resulting in a massive performance improvements.Applications where the response time is critical such as those running mobile advertising networks or telecommunication networking equipment often use IMDB's. IMDB has also gained momentum in the data analytics field that requires dynamic on-demand decision making capabilities. Table 2 summarizes the key factors that facilitate the implementation of an IMDB [3], [5], [6].

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

Table2. Drivers for IMDB implementation

Sl. No	Driver	Description
1.	Non-volatile RAM	A non-volatile battery backed up RAM ensures that the critical database information is not lost in case of power outages.
2.	Maim Memory Capacity	The ever increasing capacity of main memory makes it possible to store the entire database in memory. Current server boards support up to 12TB of RAM sizes.
3.	Multi-Core Architectures	Current systems allow up to 8-16 cores per chip. This promotes massive parallel processing and ensures rapid response to queries.
4.	Network Speed	With the arrival of faster Ethernet and Infiniband connectivity, it's possible to ensure faster communication in distributed server architecture for an in-memory database.
5.	Declining Costs	The hardware costs for storage, networking connectivity and CPU's are rapidly declining. This promotes the development of a parallelized in memory database.

II. RELATED WORK

In [7], block based compression techniques for large statistical row based databases are discussed. The primary one among them is the bit compression technique where blocks of strings can be encoded using a stream of bits. Another technique discussed is Adaptive Text Substitution (ATS) where a certain recurring pattern of characters is replaced by a pointer to the previous occurrence of this pattern.In [8], the authors propose database compression using data mining methods. This technique requires only partial decompression for read operations and no decompression for write operations. It uses the Apriori algorithm to find rules in the databases and uses these rules to store data thereby achieving high compression rates.

In [9], authors discuss various compression techniques for columnar databases. The null suppression technique facilitates compression by removing a sequence of NULL or default values and maintaining the information about the original positions of the deleted patterns in a separate data structure. Dictionary encoding is yet another compression technique in [9] where the number of bits required to represent the values of a column and the corresponding attribute values in the column are then replaced by bit encoded fixed length values. The run length encoding technique mentioned in [9] is about compressing runs of the same value in a column to a singular representation. These runs are replaced by triples of value, start position and run length to keep a track of the patterns that were compressed. This technique works best when the entries of a column are sorted so that similar valuescan appear together.

III. IN MEMORY DATABASE ARCHITECTURE

A typical IMDB has the actual database completely resident in its main memory (RAM) [10]. Thus the internal optimization algorithms are simpler and execute only a few CPU instructions. Fig. 1 illustrates a simplified architecture of a typical IMDB [7].

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

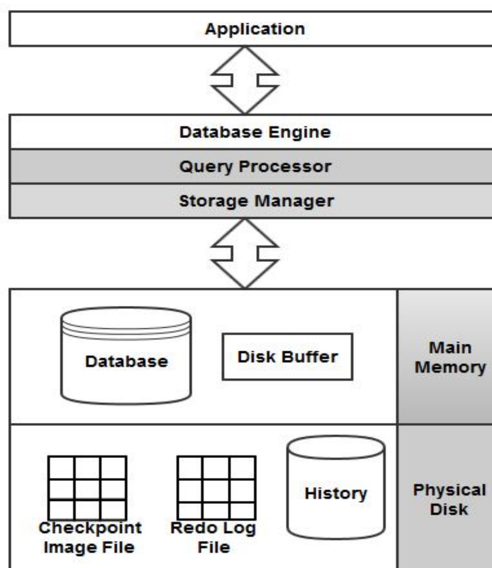

Fig. 1 IMDB architecture

Applications such as an ERP system forward the query request to an in memory query processor that resolves the query and executes it on a memory resident database [11]. In most cases, the database system uses a stored procedure which is a subroutine used within the database system to consolidate and centralize logic that was originally used in applications. This facilitates fewer data transfers and reduces layer switching between the application and the actual database since stored procedures are executed within the database system itself thereby allowing faster responses [12]. All operations executed on the IMDB are logged into a redo log file that resides on a persistent storage device typically a disk. In case the database system uses a volatile DRAM, then a power outage might cause information loss. In these scenarios, the log file that maintains a history of transactions performed is scanned and the required set of transactions are executed from a specific checkpoint time. The checkpoint image file maintains regular snapshots of the database after certain predefined intervals [13]. Further, all information present in the database is written back to the disk for backup, recovery and archival reasons. The working information of the database is present in-memory while that of older transactions such as completed shipments, processed orders is archived onto disks. The snapshot file together with the disk archival mechanism facilitates a timeless travel of the database and this archived information can be used for auditing purposes [14].

A. *Dictionary and Attribute Vector*

Although the capacity of main memory is steadily rising, RAM sizes are still restricted in comparison to disk capacities that run into several hundred terabytes of space. Hence, it's essential to use the RAM space in an efficient manner. Thus, rather than storing actual strings in tuples, the attribute column can be split into two different components namely a dictionary and an attribute vector (AV)[1]-[3]. The dictionary contains the actual string values and their corresponding implicit value identifiers (ID's). The value ID is an integer which is then stored in an attribute vector. Thus the tuple of a database row will now have an integer instead of a string. While accessing a particular field, we need to follow a two-step process. First, the integer corresponding to the value ID is accessed and then this value ID is used as a hash into the dictionary to access the actual string value. Since value ID's are integers, they can be bit encoded thereby achieving significant savings in space[14].

For example, if string for an attribute field country is "India", then rather than storing the string "India" its corresponding value ID say 91 can be stored in the tuple. If the string "India" has to be accessed, then we first access the value ID 91 and then use this ID to hash into the dictionary to access the string "India". Fig. 2 below illustrates how a column first name has been split into its corresponding dictionary and AV [3], [9],[14].

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

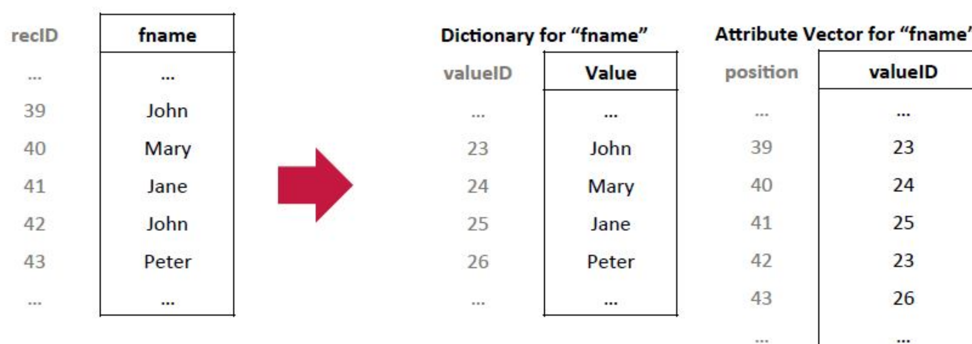

Fig. 2 Dictionary and AV for first name

Splitting a table column into a dictionary and an attribute vector allows us to store attributes as bit encoded integer values. This saves space and helps achieve the desired tuple compression while permitting direct access to the DB. Further, the dictionary for a given column can be sorted to allow rapid in memory sequential access thereby greatly improving scan performance [13]-[14].

B. *Basic Query Executions on an IMDB*

The division of a column into a dictionary and an AV changes the way in which basic query operations such as insert, delete and update can be performed on the tables [14]. For an IMDB, the insert operation will involve an initial lookup into the dictionary to check if the required value is already present [15]. If yes, then only the corresponding bit encoded value ID is directly stored in the AV. If the given value is not found in the dictionary, then we first need to insert the desired value into the dictionary, sort the dictionary and then update the AV accordingly. An update operation is performed in a manner similar to an insert operation except that an existing AV needs to be updated [15]. However, the steps involving dictionary scan, adding the new entry to the dictionary if not found and resorting the resultant dictionary remain the same. An IMDB generally doesn't perform a physical delete i.e. a delete query does not physically delete a tuple from the database. Instead, the given tuple is invalidated by using a flag. Typical flags include a timestamp which indicates the duration for which the given tuple is valid or a simple Boolean value that specifies whether a given tuple is valid or not. This operation is called a logical delete [11]. Retaining invalidated tuples allows us to perform time travels that can be used for auditing purposes. Moreover, this maintains the history of the DB by default thereby facilitating implicit logging and snapshot recording [3], [14].

IV. ATTRIBUTE VECTOR COMPRESSION

Since the AV values in an IMDB are bit encoded integers, it's possible to compress the AV further by offsetting the fixed length bit encoded data [14]. This means that a particular recurring bit pattern can be replaced by a unique value and the corresponding offset at which this replacement took place can be recorded in a bit vector. Thus, if a repeating value is replaced by a single instance of the value, a high level of compression can be achieved. There are several compression techniques available to encode the AV's some of which have been described below [1], [2], [3], [14].

A. *Prefix Encoding:*

This technique is generally used if a given column starts with a long sequence of the same value. In such cases, there's one predominant value at the beginning of a column while the remaining values are generally unique or have a low redundancy. So the recurring prefix value is then replaced by a single occurrence of the value thereby reducing the AV size significantly. This is particularly true if the data is sorted which results in multiple occurrences of the same value to occur as a prefix which results in a significant compression of the AV. This is illustrated in Fig. 3 below.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

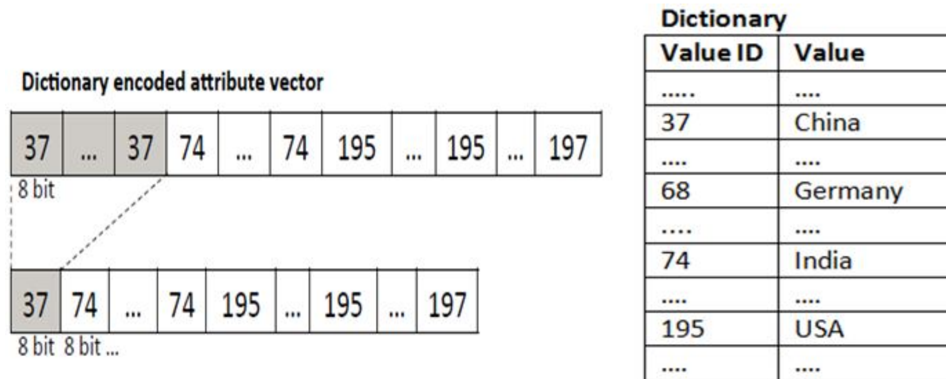

Fig. 3 Prefix Encoding

The above example shows the dictionary and AV for the column country in a sorted form. Since there are approximately 200 countries, about 8 bits are sufficient to encode this column. If the country code for China which is 37 is considered a prefix, then for the entire population of China (approx. 1.4 Billion), the country codes for citizens of China can be compressed and represented using a single common value. Since prefix encoding doesn't require any additional decompression while accessing the compressed data, it allows direct access to the compressed data [16]-[17].

B. *Run Length Encoding:*

Run length encoding is an extension of prefix encoding and can be applied when there are multiple sequences of repeating values anywhere across the column. This technique works by replacing each repeating sequence with a single occurrence in the sequence and its start position within the given column AV [14]. This is illustrated in Fig. 4 below.

Run length encoding can be generally applied for a sorted attribute vector column which results in several identical values to appear in a block. These blocks of identical values can then be replaced with a single instance of the value in the block. In the example in Fig. 4, the country codes in a country AV are presented in a sorted form. Thus codes belonging to each country appear as a block which can be encoded using run length encoding. Also, the start position of each block in the original AV is stored in a start position vector. Since there are approximately 200 countries, an 8 bit encoded value is sufficient to cover all countries. If we assume the population of China to be 1.4 Billion, then 1.4 billion country codes in a population table can be replaced by a single value thereby greatly compressing the country AV. Since the positions of the compressed sequences are known in advance, the compressed vector need not be decompressed during scans thereby promoting direct access to data[9], [18].

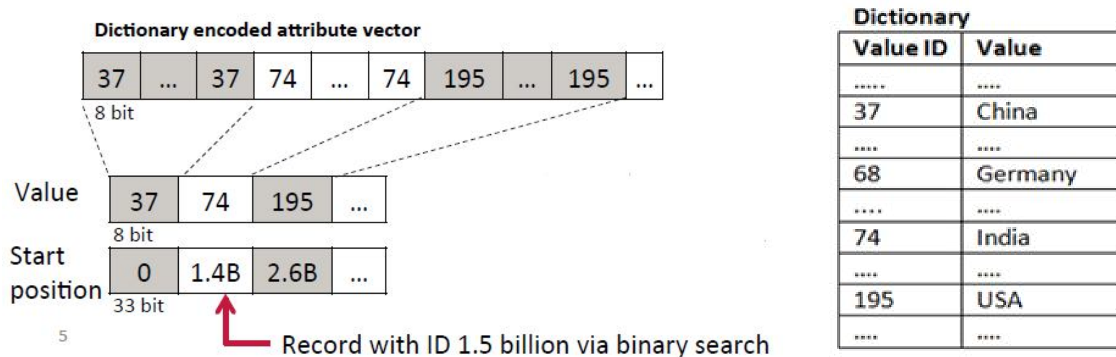

Fig.4 Run Length Encoding

C. *Cluster Encoding:*

In this technique, the given AV is partitioned into a specific number of blocks of fixed size. The typical sizeof a block is 1024 values per block. Each such block is called a cluster. In cluster encoding, if a specific cluster contains a single value then it is replaced by a single occurrence of this value [14]. A bit vector is used to indicate which clusters

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

were compressed by replacement. The size of the bit vector is equal to the number of clusters so that one bit can be used to encode a single cluster. This is illustrated in Fig. 5 below.

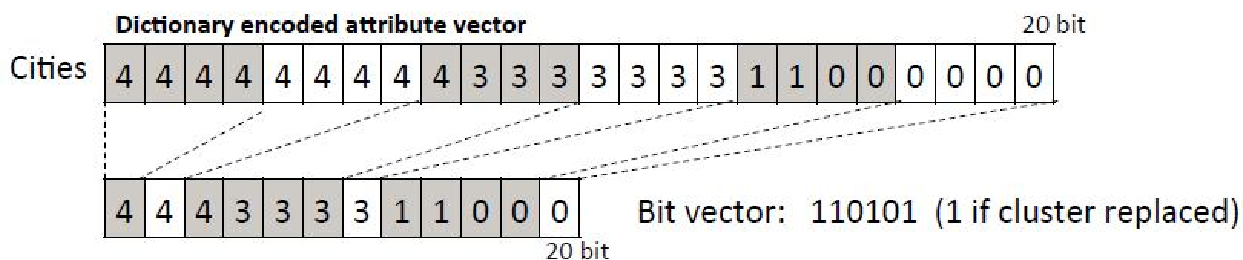

Fig. 5 Cluster encoding

The above illustration is for a city column AV. In this case, the city AV is divided into blocks of size 4 integers. In the first cluster, we have a single recurring value of 4 which is thus replaced by a single value i.e. 4. This is indicated by an entry 1 in the corresponding bit vector. On the other hand, the third sequence 4333 doesn't have a single unique value and is thus retained in its original form. This is indicated by an entry 0 in the bit vector. Thus the bit vector entry is necessary to identify the exact location in the original AV. Hence cluster encoding requires additional computations thereby allowing only an indirect access of the original AV [19]. This results in a degradation of access performance.

D. *Sparse Encoding*:

Sparse encoding involves removal of a constantly recurring value in an AV. A bit vector is used to record the position at which the given value was removed from the original AV [21]. This is particularly true in case of default or NULL values which are generally encoded as integer zero in the AV. Rather than storing millions of NULL values in an AV and wasting space, these values can be removed from the AV [21]. The resultant sparse AV will have only non-default values while the positions of removed values are maintained in the bit vector. Fig. 6 illustrates sparse encoding applied to a second nationality column AV in a population table. Since it's assumed that most people do not have a second nationality, this entry is zero in most tuples and can be removed using sparse encoding technique. Accessing a value in the compressed AV is indirect since the original position must be recomputed using the bit vector. This might have an adverse impact on scan performance [14],[20], [21].

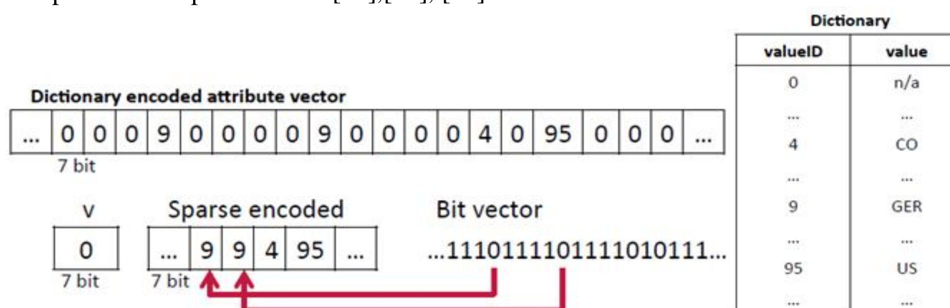

Fig. 6 Sparse encoding

E. *Indirect Encoding:*

Indirect encoding is similar to the cluster encoding technique in that both techniques split the column AV into a specified number of blocks of a fixed size (typically 1024). Each such block is called a cluster. If a cluster has only a few unique values, then these values within the cluster can be further encoded using a second level dictionary. This requires a new data buffer to store links to the new dictionaries for corresponding clusters and a bit vector to indicate as to which particular cluster is using an additional dictionary. An example of indirect encoding applied to a firstname column that has been sorted country wise is illustrated in Fig. 7 below [20]-[21]. Since block 1 below has a few distinct values, it's possible to encode block 1 using a dictionary. But in block 2 almost all values are unique so an indirect encoding using a dictionary is undesirable since it will create too many dictionary entries thereby greatly affecting the scan performance. Thus indirect encoding must be used on a block only if it has a limited number of unique values.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

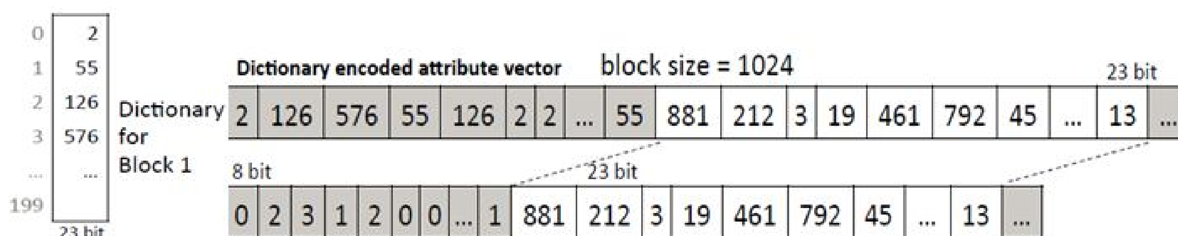

Fig. 7 Indirect encoding

V. DICTIONARY COMPRESSION

A typical dictionary in an IMDB generally consists of a value ID and its corresponding value which in most cases is a string [3]. These string values can be encoded using a standard encoding technique called delta encoding which is illustrated in Fig. 8 below [21].

Delta encoding is generally applied for sorted string values such as a dictionary of values representing the names of cities all over the world as shown in Fig. 8. Each subsequent string in the dictionary is encoded differentially by accounting for consecutive prefix characters that are common between the current string and its predecessor. The common characters are not repeated in the new string. Instead an integer value is used to encode the number of characters that are common between the prefixes of both the strings [22]. This technique promotes direct access since both the number of characters common with the precursor as well the number of characters unique to the current string are encoded within the dictionary vector itself.

If data types other than strings are present in a given dictionary, then all the values in the dictionary should be stored as sorted arrays. Once we have a sorted dictionary with a non-string data type, we can apply the same compression techniques that we applied for AV’s in case the data type is numeric [23]. In any case, a sorted dictionary will allow direct access of data and this greatly increasesthe performance of select query processing.

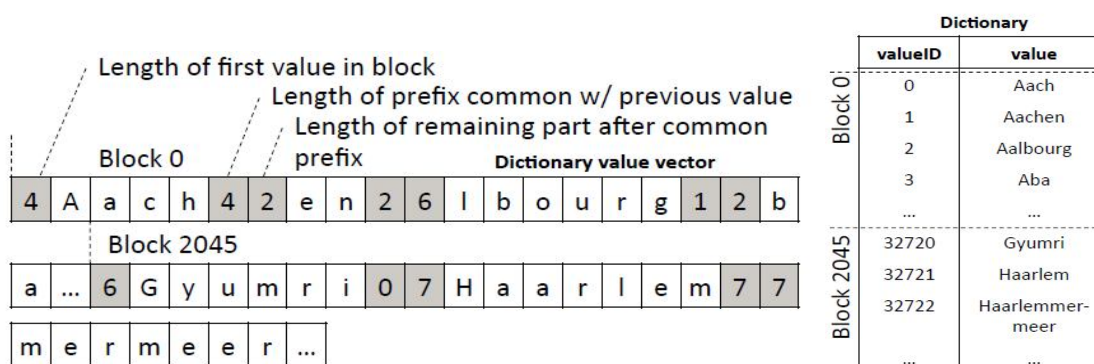

Fig. 8 Delta Encoding

VI. COMPARISON BETWEEN DIFFERENT COMPRESSION TECHNIQUES

Given the myriad number of compression techniques available for effective compression of an IMDB, the choice of the right compression technique depends on the kind of the data stored in the database and the relative arrangement of data in a column [21]. Since the data in a typical IMDB is generally sorted, any of the above listed compression techniques can be used. However, it’s always desirable that the resultant vector post compression should allow direct access to data without any additional computations otherwise it can affect the scan performance [21]. Table 3 below draws a comparison between different compression techniques that are applied to the dictionaries and AV’s of table columns [16] – [23].

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

Table3. Comparison of different compression types

Sl.No	Compression Technique	Description	Data structure compressed	Access Type
1.	Prefix encoding	A large prefix sequence replaced with a single occurrence of the value in the sequence	Attribute Vector	Direct
2.	Run length encoding	Each of the recurring sequences are replacing by a single occurrence of the value in the sequence with start positions maintained in a separate vector	Attribute Vector	Direct
3.	Cluster Encoding	The AV is split into clusters of size 1024. If all values in the cluster are same, then they are replaced by a single occurrence of the value.	Attribute Vector	Indirect
4.	Sparse Vector	Repeating values such as NULL or default values represented as zeroes in tuples are removed altogether from columns with a bit vector used to note these positions.	Attribute Vector	Indirect
5.	Indirect Encoding	If only a few entries in an AV are unique, they can be further encoded using a second level dictionary.	Attribute Vector	Direct
6.	Delta Encoding	If the data type of values in a dictionary is a string, then they can be encoded as a delta prefix of the string w.r.t its precursor	Dictionary	Direct
7.	Sorted Arrays	If the data type of values in a dictionary is a non-string, then the dictionary is stored as a sorted array for sequential direct scans.	Dictionary	Direct

VII. CONCLUSION

With the advent of massive parallel processing architectures and non-volatile batterybacked up random access memory, it has become possible to realize a database that is stored completely in memory. The ever increasing size of the main memory together with it's falling costs further provide the desired impetus for the development of an in memory database. Such a database maintains the entire enterprise data in memory while using the disk only for backup and archival purposes. Despite the increasing main memory sizes, the overall size of the main memory is still considerably smaller in comparison to a hard disk. Thus it's necessary to compress the in memory database in a way that it not only saves space but also provides sequential direct access to the compressed data. This is accomplished by splitting the database columns into dictionaries and attribute vectors. Attribute vectors can be further compressed using various techniques such as prefix encoding, run length encoding, cluster encoding, sparse encoding and indirect encoding. Dictionaries containing string values can be encoded using a technique called delta encoding. The choice of the right compression technique depends on the type of data and the relative arrangement of data within columns. If the columns are sorted, then based on the data sequences observed, appropriate compression techniques can be applied to both attribute vectors and dictionaries while still maintaining direct access to compressed data. A direct access is necessary to ensure that scan performance is not adversely impacted by any compression algorithms applied to the IMDB.

REFERENCES

1. Hasso Plattner, "A common database approach for OLTP and OLAP using an in-memory column database", In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, pp. 1-2, ACM, 2009.
2. Hasso Plattner and Alexander Zeier, In-Memory data management: Technology and applications", Springer Science & Business Media, 2012.
3. Hasso Plattner, "A course in in-memory data management", Berlin/New York: Springer, 2013.
4. "Definition: in-memory database", Sept. 2013, [Online]. Available: http://whatis.techtarget.com/definition/in-memory-database.
5. Michael Vizard, "The Rise of In-Memory Databases", July 2012, [Online]. Available: http://insights.dice.com/2012/07/13/the-rise-of-in-memory-databases/.
6. Irfan Khan, "Falling RAM Prices Drive In-Memory Database Surge", SAP, Oct. 2012, [Online]. Available: http://www.digitalistmag.com/big-data/ram-prices-drive-in-memory-surge-020097.
7. Wee Keong Ng and Chinya V. Ravishankar, "Block-oriented compression techniques for large statistical databases", In IEEE Transactions on Knowledge and Data Engineering, vol. 9, no. 2 pp.314-328, 1997.
8. Chien-Le Goh, Kazuki Aisaka, Masahiko Tsukamoto and Shojiro Nishio, "Database compression with data mining methods", In Information organization and databases, pp. 177-190. Springer US, 2000.
9. Daniel Abadi, Samuel Madden and Miguel Ferreira, "Integrating compression and execution in column-oriented database systems", In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pp. 671-682, ACM, 2006.
10. Garcia-Molina, Hector, and Kenneth Salem, "Main memory database systems: An overview." In IEEE Transactions on Knowledge and Data Engineering, vol. 4, no. 6, pp. 509-516, 1992.
11. Brain Berkowitz, Sreenivas Simhadri, Peter A. Christofferson and Gunnar Mein, "In-memory database system", U.S. Patent 6,457,021, issued September 24, 2002.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 9, September 2015

12. Joanes Bomfim and Richard Rothstein, "In-memory database for high performance, parallel transaction processing", U.S. Patent Application 10/193,672, filed July 12, 2002.
13. "Oracle TimesTen In-Memory Database Architectural Overview", v6.0, [Online]. Available: http://download.oracle.com/otn_hosted_doc/timesten/603/TimesTen-Documentation/arch.pdf.
14. Hasso Plattner, "SanssouciDB: An In-Memory Database for Processing Enterprise Workloads", In BTW, vol. 20, pp. 2-21, 2011.
15. Tobin Lehman and Michael J. Carey, "Query processing in main memory database management systems", vol. 15, no. 2, ACM, 1986.
16. "Computer Algorithms: Data Compression with Prefix Encoding", Stoimen's Web Blog, Feb. 2012, [Online]. Available: http://www.stoimen.com/blog/2012/02/06/computer-algorithms-data-compression-with-prefix-encoding/.
17. Toby Berger, Yi-Jen Chiu and Mikio Kawahara, "Byte-based prefix encoding", U.S. Patent 5,973,626, issued October 26, 1999.
18. Victor Watson, "Run-length encoding", U.S. Patent Application 10/143,542, filed May 10, 2002.
19. Zan Ouyang, Nasir Memon, Torsten Suel and Dimitre Trendafilov, "Cluster-based delta compression of a collection of files", In Proceedings of the Third International Conference on Web Information Systems Engineering, pp. 257-266. IEEE, 2002.
20. "Data compression techniques for performance improvement of memory-intensive applications on shared memory architectures", Apr. 2010, [Online]. Available: http://people.inf.ethz.ch/akourtis/phd/phd-en.pdf.
21. Hasso Plattner and Alexander Zeier "In-Memory data management: Technology and applications", Springer Science & Business Media, 2012.
22. Jeffrey Mogul, Fred Douglis, Anja Feldmann and Balachander Krishnamurthy, "Potential benefits of delta encoding and data compression for HTTP", In ACM SIGCOMM Computer Communication Review, vol. 27, no. 4, pp. 181-194. ACM, 1997.
23. Long Cheng, A. Malik, S. Kotoulas, T. E. Ward and G. Theodoropoulos, "Efficient parallel dictionary encoding for RDF data", In Proceedings of the 17th International Workshop on the Web and Databases, WebDB, vol. 14, 2014.