http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (1): 160-164, 2008
ISSN 1812-5638

An Efficient Multi-Component Indexing Embedded Bitmap Compression for Data Reorganization

Yashvardhan Sharma and Navneet Goyal
Department of Computer Science and Information Systems,
Birla Institute of Technology and Science, Pilani, Rajasthan, 333031, India

Abstract: In the present study, we discuss bitmap indices with compression using multi-component indexing for the efficient storage and fast retrieval of large scientific data. The bitmap compression indices embedded multi-component shows superiority over bitmap compressed indices. Gray Code ordering algorithm is used which runs in linear time in the order of the size of the database. Reduction in the number of columns is observed when multi-component indexing is applied on the binned data. An improvement in the space requirement for Bitmap Index by 25% is observed when one time component indexing is applied. Satisfactory improvement factor is observed when gray code ordering and WAH compression technique is used. Due to processing overhead, two component indexes is used. Tuple reordering problem is studied to reorganize database tuples for optimal compression rates. The experimental results on real data sets show that the compression ratio shows the improvement by a factor of 2 to 8.

Key words: Bitmap index, multi-component index, compression, data warehouse

INTRODUCTION

Data warehouses and Scientific Databases contain data consolidated from several operational databases and provide the historical and summarized data. On-Line Analytical Processing (OLAP) provides advanced analysis tools to extract information. Fast response time is essential for on-line decision support system. A promising approach to process complex queries in Decision Support Systems (DSS) is the use of bitmap indexing (O'Neil and Quass, 1997). Bitmap indexes have been implemented in several commercial DBMSs as IBM, Informix, Oracle, Red Brick and Sybase. This is an indication that the bitmap indexing technique is indeed an efficient and practical. A major advantage of bitmap indexes is that bitmap manipulations using bit-wise operators (AND, OR, XOR, NOT) are efficiently supported by hardware. Moreover, bitmap indexes are space efficient, especially for attributes with low cardinality. The basic bitmap index scheme builds one bitmap for each distinct value of the attribute indexed and each bitmap has as many bits as the number of tuples.

The size of such index can be very large for high cardinality attribute where there are thousands or even millions of distinct values. Many strategies have been devised to reduce the index sizes, such as, more compact encoding strategies (Chan and Ioannidis, 1998, 1999) binning and compression (Antoshenkov, 1994; Wu *et al*., 2001, 2002; Johnson, 1999). (Wu *et al*., 2001) has discussed some of empirical studies and shown that some compression schemes can reduce the index size as well as the query response time. Some general purpose text compression schemes, such as LZ77, can compress bitmaps. However these schemes are not efficient for answering queries (Wu *et al*., 2001; Johnson, 1999). Bit-wise logical operations on bitmaps compressed with a typical text compression algorithm are generally much slower than the same operations on the uncompressed bitmaps. To improve the speed of operations, a number of specialized bitmap compression schemes are the Byte-aligned Bitmap Code (BBC) and the Word-Aligned Hybrid code (WAH). Both are based on the run-length encoding. They represent a long sequence of 0s or 1s using a counter and represent a mixture of 0s and 1s literally.

Bitmap Compression may not be enough for the enormous data generated in some applications such as high-energy physics. To improve the compression rates, reorganization of bitmap tables is discussed by Pinar *et al*. (2005), where tuple reordering problem is introduced through gray code ordering algorithm. The

Corresponding Author: Yashvardhan Sharma, Department of Computer Science and Information Systems,
Birla Institute of Technology and Science, Pilani, Rajasthan, 333031, India

formulation concern is to reduce the number of columns for binned bitmap tables to improve better ordering and improve the space complexity for tuple reordering problem. In tuple reordering problem, we found that for lesser number of columns, gray code ordering gives better compression rates. We have applied Multi-Component indexing technique to reduce the number of columns on binned bitmap tables. We have observed 25% reduction in compress file size on various real time applications data sets.

Variants of bitmap indexes

Bitmap index: A bitmap index for a desired attribute of a database table is a collection of bit-vectors of length equal to the number of records in the database table, one for each possible value that may appear in the desired attribute. The vector for value v has 1 in position i if the ith record has v in the desired attribute and it has 0 otherwise. This bitmap organization matrix is also called a value-list index, because each 1 in it represents a specific value. If a query now wants to find all records with specific values for the attribute, it can select them by using boolean operators.

Multi-component indexes: Simple bitmap indexes take huge amount of space for high cardinality data since we need to make bitmap vector for each distinct value. By using Multi-component Indexes, number of bitmap vectors can be reduced. The general idea behind multi-component index is to perform the Attribute Value Decomposition (AVD). One value can be decomposed into several components with same base or different bases. Instead of representing bitmap with single table, the same values can be represented with several smaller Bitmaps working together. Let C be the attribute cardinality, which means the number of actual values that an attribute can have. Then a bitmap index can be created in the following way.

Consider an attribute value *v* and a sequence of (n-1) numbers $\langle b_{n-1}, b_{n-2},...,b_1 \rangle$.

Let us define $b_n = \left\lceil \frac{C}{\prod_{j=1}^{n-1} b_j} \right\rceil$.

Further, decompose v into a sequence of n digits $\langle v_n, v_{n-1},...,v_1 \rangle$. as follows:

$$\begin{aligned} v &= V_1 \\ &= V_2 b_1 + v_1 \\ &= V_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= V_4(b_4 b_2 b_1) + v_3(b_2 b_1) + v_2 b_1 + v_1 \\ &\vdots \\ &= v_n\left(\prod_{j=1}^{n-1} b_j\right) + ... + v_i\left(\prod_{j=1}^{i-1} b_j\right) + ... + v_2 b_1 + v_1 \end{aligned}$$

Where:

$$v_i = V_i \bmod b_i, V_i = \left\lfloor \frac{V_{i-1}}{b_{i-1}} \right\rfloor, 1 < i \le n, 0 \le v_i < b_i$$

Based on the above equation, each choice of n and sequence $\langle b_{n-1},b_{n-2},...,b_1 \rangle$, gives a different representation of attribute values and therefore a different index. The index consists of n components, i.e., one component per digit. Each component individually is a collection of bitmaps, constituting essentially a base- b_j index. The sequence $\langle b_{n-1}, b_{n-2},...,b_1 \rangle$ is called a base-$\langle b_{n-1}, b_{n-2},...,b_1 \rangle$ index. The index consists of n components, which means that each choice of n and a sequence of $\langle b_{n-1}, b_{n-2},...,b_1 \rangle$ give a different representation of the index because each component individually is now a collection of bitmaps (Chan and Ioannnidis, 1998). Lets take an example of a 2-component encoded bitmap index for an attribute with cardinality c = 1000, the two components have base sizes of b_1 = 25 and b_2 = 40. Assume the attribute values are in the domain of (0; 999). An attribute value *v* is decomposed into two components with c_1 = v/40 and c_2 = v40%. The component c_1 can be treated as an integer attribute in the range of 0 and 24; the component c_2 can be viewed as an integer attribute in the range of 0 and 39.

Binning: High Cardinality data is divided into number of bins to reduce the number the bitmap vectors for each attribute. The basic idea of binning is to build a bitmap for a bin rather than each distinct attribute value. This strategy disassociates the number of bitmaps from the attribute cardinality and allows one to build a bitmap index of a prescribed size, no matter how large the attribute cardinality is. A clear advantage of this approach is that it allows one to control the index size. If a value falls into a bin, this bin is marked 1 otherwise 0. Since a value can only fall into a single bin, only a single 1 can exist for each row of each attribute. After binning, the whole database is converted into a huge 0-1 bitmap, where rows correspond to tuples and columns correspond to bin.

WAH compression technique: This scheme is a variation on the run-length code. The essence of the run-length

code is to represent a list of consecutive identical bits by its length and its bit value. In 32-bit implementation, the Leftmost Bit (LMB) of a word is used to distinguish between a literal word and a fill word, where 0 indicates a literal word and 1 indicates a fill word. The lower 31 bits of a literal word contains literal bit values. The second leftmost bit of a fill word is the fill bit and the 30 lower bits store the fill length. To achieve fast operation, it is crucial to impose the word-alignment requirement on this scheme. The word-alignment requirement in WAH requires all fill lengths to be integer multiples of 31 bits (i.e., literal word size). Given this restriction, we represent fill lengths in multiples of literal word size. For example, if a fill contains 62 bits, the fill length will be recorded as two (Wu *et al*., 2001; Johnson, 1999) (Fig. 1).

Tuple reordering problem: Tuple Reordering Problem aims at reordering the database tuples to increase the performance of run-length encoding by having longer uniform segments and fewer number of blocks. To reorder large database is a big challenge due to their large size. Run-encoding packs each segment of 1 s into a block and stores a pointer to each block together with the length of the block. Thus the storage size is determined by the number of such blocks. Consider two consecutive tuples in the bitmap table. If the tuples are on the same bin for an attribute, they will be packed to the same block. If not, a new block should start. Efficiency can be enhanced by reordering tuples so that they fall into the same bins as much as possible.

Gray code ordering: A gray code is an encoding of numbers so that adjacent numbers have only a single digit differing by 1. For binary numbers two adjacent numbers differ only by one digit. For instance (000, 001, 011, 010, 110, 111, 101, 100) is a binary gray code. Binary gray code is often referred to as the reflected code, because it can be generated by the reflection technique described below.

- Let S = (s_1, s_2,...s_n) be a gray code.
- First write *S* forwards and then append the same code *S* by writing it backwards, so that we have (s_1, s_2,...s_n, s_n, s_{n-1},...,s_1).
- Append 0 at the beginning of the first n numbers and 1 at the beginning of the last n numbers. Gray code sorting starts with dividing numbers that start with 0 and those that start with 1.

Clearly those that start with 0 will precede others in the ordering. Then we can recursively order those that start with 0. The same can be applied to the second group of numbers that start with 1, but we need to reverse their ordering due to the reflective property of the Gray code.

Experiment and comparative illustrations: We present the results of experiments on real application datasets. The experiments were conducted on a Pentium 4 machine with 1 Gb RAM running Linux. We compare here three preprocessing schemes on bitmap data, before actually applying WAH Compression algorithm. Since, the warehoused data is read mostly, the time involved in preprocessing is not a major concern here. We took the uncompressed data with bin size 8, in the first scheme, applied multi-component indexing, there by reducing the bin size from 8 to 6. In the second scheme, we applied multi-component indexing twice, in a single step, reducing the bin size from 8 to 3. And in third scheme, we applied multi-component indexing, but saved the two components in two different files, in order to see the effect on Compression algorithm. We have collected the results by

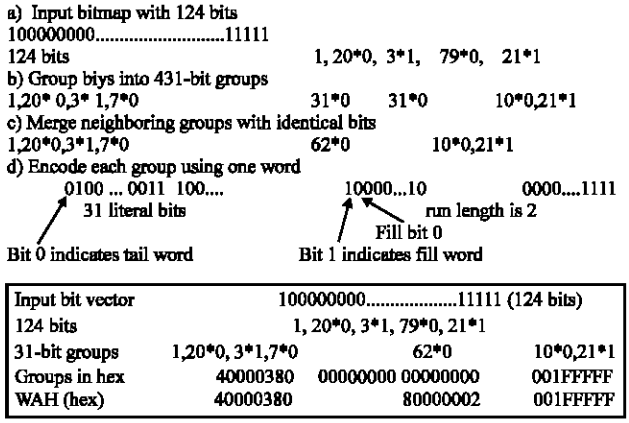

Fig. 1: WAH bit vector

Table 1: Improvement in compression of real data sets

Name	Uncompressed size (bytes) original	Uncompressed size (bytes) multi component	Compressed size (bytes) original	Compressed size (bytes) reordered	Improvement factor
Histo 64	6208839	4659655	267399	135558	1.972580
Gaussian 16	12900000	9700000	2786733	1965789	1.417616
Stock 360	18726500	14046500	1448424	196461	7.372578
Histobig 64	57641193	43258985	1935619	497493	3.890746
Stockdft 360	18726500	10858304	1172268	612837	1.912854
Histobigsvd 64	57641193	43258985	4011647	1996345	2.009496

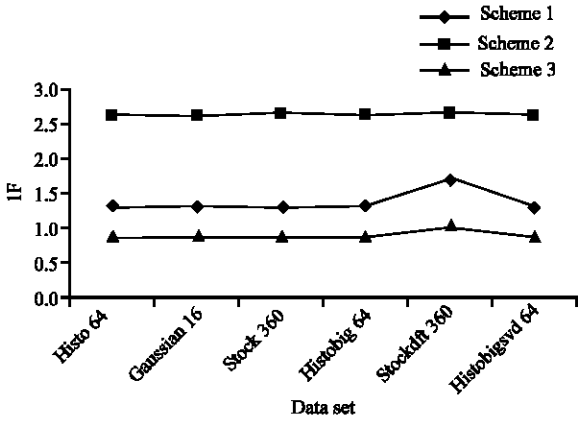

Fig. 2: Double multi-component on all three schemes

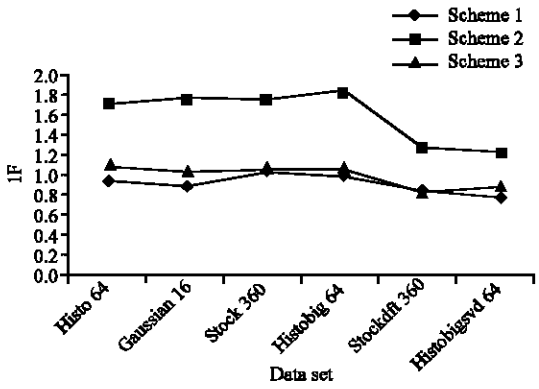

Fig. 3: Effect of preprocessing schemes with WAH compression

looking at the improvement in storage of datasets. The Improvement Factor (IF) is defined as

$$\text{Improvement Factor (IF)} = \frac{\text{Origional size}}{\text{Compressed size}}$$

That is the ratio of original file size to compressed file size. The Improvement Factor achieved by applying multi component indexing is a straight forward 1.33 for multi component indexing, as the bin size is initially 8 while multi component indexing reduces it to 6. The case for double multi component indexing is 2.66, it reduces bin size to 3 (Fig. 2).

Table 1 shows the effectiveness of our approach on six data sets from various applications. In this table, the first two columns give sizes of the uncompressed bitmap tables for the original and multicomponented data and next two columns give sizes of the compressed bitmap tables for the original and multicomponented data. The last column presents the improvement factor. The data set, histogram, comes from an image database with 112,361 images. Images are collected from a commercial CD-ROM and 64-dimensional color histograms are computed as feature vectors. The data set, stock, is a time series data which contains 360 days stock price movements of 6500 companies, i.e., 6500 data points with dimensionality 360. The data set histogram is partially correlated, whereas the stock data set is highly correlated.

The second set of experiments deals with the performance achieved by WAH Compression algorithm when either of schemes one, two or three is used as a preprocessing step before applying WAH. Here improvement factors have been calculated using the file size obtained by applying WAH on dataset and by applying one of the schemes and then applying WAH to it. This allows for comparison of the effect of each of the schemes on the WAH compression. The second scheme out performs the other two, which is obvious. The more interesting point in these results is that storing in two different files may not affect the compression of WAH. Table 1 shows the result of 25 % space reduction and improvement factors.

The third set of experiments deals with the effect of the three preprocessing schemes on re-ordering tuples before applying the compression algorithm. As with results above, the improvement factors here are calculated with respect to running the re-ordering and compression algorithms directly on datasets. Ignoring the obvious good results of scheme 2, in the Fig. 3 shows scheme 3 out performs scheme 1. This is because the reordering algorithm is better able to align the tuples for better compression.

From the experiments, it shows that multi component indexing improves the performance of compression algorithms providing for better storage of warehoused

bitmap data. Choosing the base for multi component indexing is critical, as shown by results of scheme 2. Moreover, it was established that storing components in different files improves storage, if tuple reordering is done before the actual compression.

CONCLUSIONS

We have described tuple reordering problem to improve bitmap compression rate for large datasets. We applied multi-component indexing to get maximum benefits of gray code sorting algorithm, which is an in-place algorithm and runs in linear time in the order of the size of the database. Multi component indexing improves the performance of compression algorithms providing for better storage of warehoused bitmap data. An improvement in the space requirement for bitmap index by 25% is observed when one time component indexing is applied. Satisfactory improvement factor is observed when gray code ordering and WAH compression technique is used. And also, it was established that storing components in different files improves storage, if tuple re-ordering is done before the actual compression. Choosing the base for multi component indexing is critical and thus finding a good base that maximizes the performance of WAH will be another interesting research project.

REFERENCES

Antoshenkov, G., 1994. Byte-aligned bitmap compression. Technical Report, Oracle Corp., US. Patent No. 5, pp: 363-398.

Chan, C.Y. and Y.E. Ioannidis, 1998. Bitmap index design and evaluation. Proceedings of the International ACM SIGMOD Conference, ACM Press, pp: 355-366.

Chan, C.Y. and Y.E. Ioannidis, 1999. An efficient bitmap encoding scheme for selection queries. Proceedings of the International ACM SIGMOD Conference, ACM Press, pp: 215-226.

Johnson, T., 1999. Performance measurements of compressed bitmap indices. Proceedings of the International Conference on Very Large Data Bases, Morgan Kaufmann, pp: 278-289.

O'Neil, P. and D. Quass, 1997. Improved query performance with variant indexes. Proceedings of the International ACM SIGMOD Conference, Arizona, pp: 38-49.

Pinar, A., T. Tao and H. Ferhatosmanoglu, 2005. Compressing bitmap indices by data reorganization. Proceeding of the International Conference on Data Engineering, pp: 183-194.

Wu, K., E.J. Otoo and A. Shoshani, 2001. A performance comparison of bitmap indexes. Proceedings of the International Conference on Information and Knowledge Management, ACM Press, pp: 559-561.

Wu, K., E.J. Otoo and A. Shoshani, 2002. Compressing bitmap indexes for faster search operations. Proceedings of the International Conference on Scientific and Statistical Database Management, pp: 99-108.