International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 2, Issue 3, March 2016

A System for Query Processing and Optimization Using Variable Length Encoding

[1] Akshita S. Khoriya [2] Madhuri S. Madeshi [3] Anand V. Saurkar [4] Payal S. Chirde [5] Shreya S. Kandekar [6] Chiranjeev D. Garhwani
[1][2][3][4][5][6] Department of Computer Science and Engineering
Datta Meghe Institute of Engineering, Technology and Research, Wardha (MS), India
[1] khoriyaakshita@gmail.com [2] madeshimadhuri@gmail.com[3]saurkaranand@gmail.com
[4]payal_chirde@rediffmail.com [5] prachi1994kandekar@gmail.com [6] chiranjeevdg@gmail.com

Abstract: **In data warehousing and OLAP applications, large amount of data are processed. To perform operations of predicates in SQL, large amount of data become highly inadequate which requires supporting to compare a tuples of group with a number of values. Currently available queries are complex, complex to write and create as well as challenging for database engine to optimize, which results in costly evaluation. Many of the available query processing algorithms does not take the advantage of the small-result-set property, which incurs intensive disk accesses as well as needed computations, which results in long processing time. Optimized query processing approach achieved by various studied algorithms shows very good performance to processing set predicates. We presented here bitmap index and developed a very effective bitmap pruning strategy by using Huffman coding which is variable length compression technique for processing queries, which completely removes the necessity of scanning and processing the entire data set (table), which results in efficient execution of the query processing. Experiments verified our technique is much more efficient than existing algorithms in processing queries and retrieving data from large datasets.**

***Keywords*— Bitmap index,Data Warehousing, Huffman Coding ,OLAP, Querying processing and optimization , VLC, Word-Aligned Hybrid(WAH).**

I. INTRODUCTION

Now a day, the demand of querying the data in data warehouse and OLAP applications with the semantics of set-level comparison is very high. Suppose a institution seeking for candidates for the job with set of compulsory skills, company or institution may search their resume database. Skill of each candidate that is set of values are compared against the compulsory skills. Such a sets are dynamically formed. Such process of set level comparisons can be performed using currently available SQL syntax and semantics without proposed system [1].If the set level comparisons are perform using currently available SQL syntax, resulting query may have more complexity, with the result it may take too much time to process the query than necessary time. Such complex query becomes a difficult for the user to formulate, which will result in too much costly evaluation.

Aggregation query is a type of Iceberg Query [3] which calculates and computes aggregate value above the particular threshold values. High aggregate values will always carry out more necessary information. The Aggregate functions are COUNT, MIN, MAX, SUM and AVERAGE. In this paper, main focus is on processing queries that has aggregation function with anti-monotone property [4] such as the MIN, MAX, SUM and COUNT.

In this paper, our aim is to process and retrieve the data using the compressed Bitmap index. Currently available GROUP BY clause can only do scalar values comparison by accompanying HAVING clause. Aggregate functions COUNT, MIN, MAX, SUM and AVERAGE produce single numeric value, which compared to another single aggregate value. We have presented the Aggregate functions based technique and compressed bitmap index based technique. Aggregate functions based technique processes set predicates in the normal way as processing the conventional aggregate function. Second technique is compressed bitmap index in which bitmap indices are created on each of the attributes. This technique will be more efficient because it focuses on only those tuples which satisfies the query condition and bitmaps of appropriate columns. Such index structure can be applicable on many different types of attributes. This technique process queries such as selections, joins, multi-attribute grouping etc [1]. For the purpose of the compression Word-Aligned Hybrid (WAH) [5] technique is used. This technique now a day can be applied on all types of the attributes such as numeric attribute[6][7] high cardinality categorical attributes[6],text attributes[8] etc.

This technique is efficient for the data warehouse query processing and OLAP [9].

II. RELATED WORK

Now a day, many database management systems provide the definition of attributes consisting a set of values such as the nested table in Oracle and SET data type in MYSQL. For the Set predicates, there is no need of data storage and representation; hence they are included in standard DBMS. In the real world applications, according to need of query group and corresponding set are usually dynamically formed. Users can dynamically form the set level comparisons without any limitations caused by the database schema for the set predicates. It also allows cross attribute set level comparison. In [10][11][12],grouping variables and the associated set concepts was introduces as SQL extension in order to allow comparison of multiple aggregate functions over the same grouping condition. This paper mainly focuses on processing of the data using compressed bitmap index and predicting the sets.

Bin He et al (2012) explained properties of bitmap index and developed a very efficient and powerful bitmap index pruning strategy for the query processing. Bitmap Index pruning based technique removes the necessity of scanning and processing of the entire data set (table) and thus results in processing of fast query processing. This technique is more efficient than the existing algorithms generally used in the recent databases. By checking these characteristics of bitmap indices, the opportunities of computing queries efficiently using compressed bitmap index. A naive way for computation of query used for the bitmap indexing is to do pairwise bitwise-AND operations among bitmap vectors of all the necessary attributes. This technique is not very efficient because the product of the number of bitmap vectors of all the attributes is large and large portion of these operations are not necessary.

Elizabeth O'Neil et al proposed the FASTBIT and RIDBIT techniques. FastBit is research tool developed for study and analyzing how the compression methods affect bitmap indexes, and has been used in a number of scientific applications [12]. It organizes the table data into rows and columns, where each table is the vertical partitioned and each column stored in an individual files, each partition typically consists of many millions of rows. Bitmap indexes are applied continuously without partitioning into bit segments as in the RIDBit technique. The index used in this study is that about the Word-aligned hybrid (WAH) compression by a basic bitmap index. In FastBit tool bitmaps generates all the values of entire indexing for one individual in the memory before writing the index file. In this section we are presenting the background on current techniques which are used to compress bitmap indices that will achieve the fast querying.

Run-length encoding schemes accomplish the compression when sequences of successive identical bits, and "runs", are presents. BBC [11] is an 8-bit hybrid RLE representation which is in the practice of a literal or fill. The MSB which is known as the flag bits mark the type of encoding. That is, a byte 0xxxxxxx which will be denoting the least significant 7 bit is a literal representation for the genuine bit string. In distinction, 1xnnnnnn encodes a fill which compactly represents the runs of consecutive x's. Here, x are the fill bit which encodes the Compressed bitmap indexes are increasingly utilised for the efficiently querying of very large databases. The Word Aligned Hybrid (WAH) bitmap compression schemes are the commonly recognized for the most efficient compression scheme in terms of the CPU efficiency. WAH [16][17], is not like BBC, that uses a 31 bits representation (32 bits including the flag bit). This representation offers the several benefits over BBC—one being used for the certain bitmaps, WAH can achieve significant speedup in the query processing time duration when it is compared to BBC. These speed ups are due to the fact that memory is naturally raised by the CPU the words at a time. By using a Word-Aligned Encoding, WAH avoids the overhead of the further extraction of bytes within a word that is incurred by the BBC. Thus, WAH not only compressed the literals more effectively than BBC (using 4 less flag bits per 31 bits), but it also can be used to practice bitwise operations more quicker over literals by avoiding the overhead of the byte abstraction or parsing and decoding to determine if the byte are indeed the literal.

B-Tree is a self-balancing search tree. In most of the self-balancing search trees like AVL and redly blackly trees, it is assuming that everything are in the main memory. To understand the usage of B-Trees, we must think of huge amount of data that cannot fit in the memory.[20] When the number of keys is high, the data is read from disk in the form of a block. Disk access time is very high as compared to main memory access time. The main idea of using B-Trees is for reducing the numbers of disk accesses. Most of the tree operations (search, insert, delete, max, minuet) requires O(h) disk accessed where h is the height of the tree. B-tree is a fat tree. Height of B-Trees is kept low by putting the maximum number of possible keys in a B-Tree node. Generally, a B-Tree node size is kept equal to the disk block size. Since h are low for B-Tree, total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees like AVL Trees, and Red Black Tree, .etc.

Properties of B tree are:- All leaves are at same level. A B-Tree is defined for the term minimum degree't'. The value of t depends upon disk block size. Every node except root should contain at least t-1 keys. Root may contain minimum 1 key. All nodes (including root) may contain at

most 2t – 1 key. Numbers of children of nodes are equal to the number of keys in it plus 1. All keys of a node is sorted in the increasing order. The children between two keys k1 and k2 contained all keys in range from k1 and k2. B-Tree grows and shrinks from root which is unlike Binary Search Tree. Binary Search tree grows downward.

The B-Tree Index is popular in applications of data warehouse for high cardinality column such as name since the space usage of the index is not dependent of the column cardinality. However, the B-Tree Indexing has characteristics that made them poor choice for DW's queries. First of all, a B-Tree index is of no use for low cardinalities data like the gender column since it reduces very few numbers of I/Os and it may uses more space and time than the raw indexed column. Second is that, each of the B-Tree Index is independent and thus could not operate with each other on an indexing level before going for the primary source. At last, the B-Tree Index fetches the results of the data ordered by key values which has unordered row ids, so more I/O operations and page faults are generated [19].

A B+ tree is a data structure used in the implementation of database indexes. Each node of tree contains an ordered list of the keys and pointers to lower level nodes in the tree. The pointers which are used can be thought of as being between each of the keys. For searching or for inserting an element into the tree, one loads up the root node, than it finds the adjacent keys and the searched for value is between, and follows the corresponding pointer to the next node in the tree. Recurring eventually leads to the desired value or the conclusion that the value is not present.

B+ trees use clever balancing techniques to make sure that all of the leaves are always on the same level of the tree, that each node is always at least half full of keys, and that the height of the tree is always at most ceiling (log(n)/log(k/2)) where n is the number of values in the tree and k is the maximum number of keys in each block. This means that only a small number of pointer traversals are necessary to search for a value if the number of keys in a node is large. This is crucial in a database because the B+ tree is on disk. Reading a single block takes just as much time as reading partial block, and a block can hold a large number of pointers.

B+ trees can also be used out region of the disk, but generally a balanced binary search tree or a skip list or something should provide better performance in memory, where pointer following are no more expensive than finding the right pointer to follow[21].

A Huffman Coding is most sophisticated and efficient lossless data compression technique. In Huffman Coding the typescripts in a data files are converted into the binary codes. And in this technique the most common characters of the file has shortest binary code, and also has the least common have the longest binary code.

III. PROPOSED METHOD

In proposed system we have presented compressed bitmap Index based technique using Word-Aligned Hybrid (WAH) compression technique and bitmap index with Variable length encoding based technique. In table R, column A has three distinct values "A1;A2;A3," and column B has three distinct values "B1;B2;B3." The bitmap indices are those on the right of Fig. 1. To process the iceberg query in Fig. 2, the naïve approach will conduct bitwise-AND operations between nine pairs: (A1, B1), (A1, B2), (A1, B3), (A2, B1), (A2, B2), (A2, B3), (A3, B1), (A3, B2), and (A3, B3). After each of the Bitwise-AND operations, number of 1 bits of the resulting bitmap vector are counted. If the number of 1 bits is larger than the threshold (2 in this example), it is added into the iceberg result set.

A	B	C	.....
A2	B2	1.23	
A1	B3	2.34	.....
A2	B1	5.56	
A2	B2	8.36	
A1	B3	3.27	
A2	B1	9.45	
A2	B2	6.23	
A2	B1	1.98	
A1	B3	8.23	
A2	B2	0.11	
A3	B1	3.44	
A3	B1	2.08	

(a) Table R

A1	A2	A3
0	1	0
1	0	0
0	1	0
0	1	0
1	0	0
0	1	0
0	1	0
0	1	0
1	0	0
0	1	0
0	0	1
0	0	1

B1	B2	B3
0	1	0
0	0	1
1	0	0
0	1	0
0	0	1
1	0	0
0	1	0
1	0	0
0	0	1
0	1	0
1	0	0
1	0	0

(b) Bitmap indices for A,B

Fig 1.An example of Bitmap index

The Word Aligned Hybrid Compression technique performs compression on Bitmap indexing which generates an extra table for further compression. This causes the space complexity. As it consumes more space, execution time also increases for retrieving the data for given query. Complex queries containing scalar-level operations are often formed to obtain even very simple set-level semantics. Such complex queries are much difficult for users to formulate. Currently available bitmap indexing approach not supported in major Database platforms such as MySQL, DB2 except Oracle. To overcome the pitfalls of existing system we are proposing new compression technique that is Variable Length Compression Technique that will improve the performance of the system that will minimize the space complexity and will also improves the execution time of query processing.

Following is the complete execution of the proposed system.

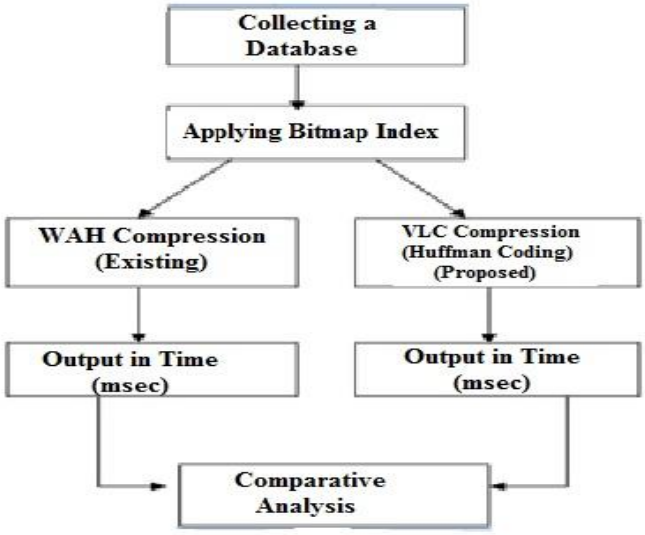

Fig 2 .A proposed System

IV. RESULT AND DISCUSSION

The experiments were performed on the Intel I5 CPU @1.90 GHz processor with 2GB RAM memory. We conducted experiments on bitmap index based technique, compressed bitmap index based technique with WAH compression and Variable Length Coding such as Huffman coding. The experiments are carried on road-accidents safety dataset taken from official web site of United Kingdom (UK).Dataset consist of road accidents dataset, casualties dataset and vehicles dataset from the year 2005 to 2012. Each table in the Road-safety dataset consist of approximately 1355615.

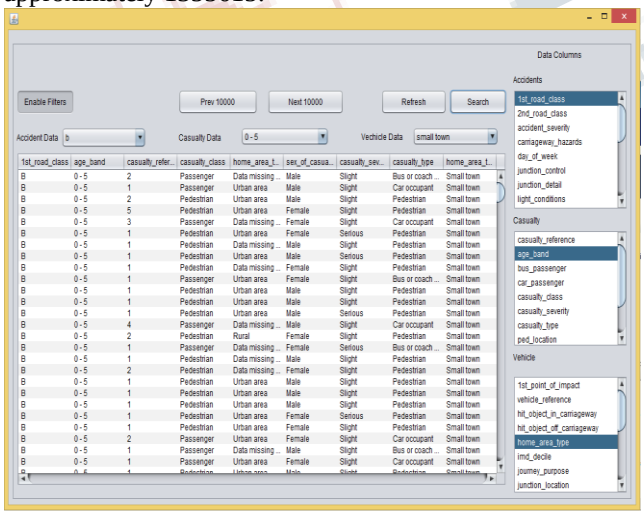

Figure 3.Records retrieved using WAH Compression Technique.[Total Execution Time =45959 ms].

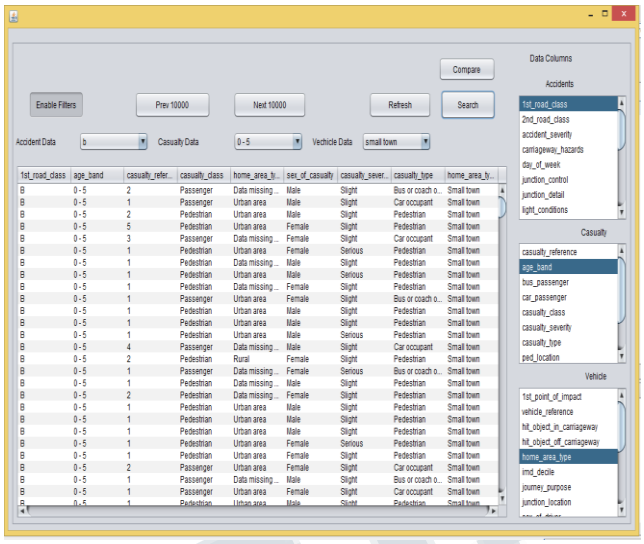

Figure 4.Records retrieved using VLC Huffman Coding Compression Technique.[Total Execution Time =38531 ms].

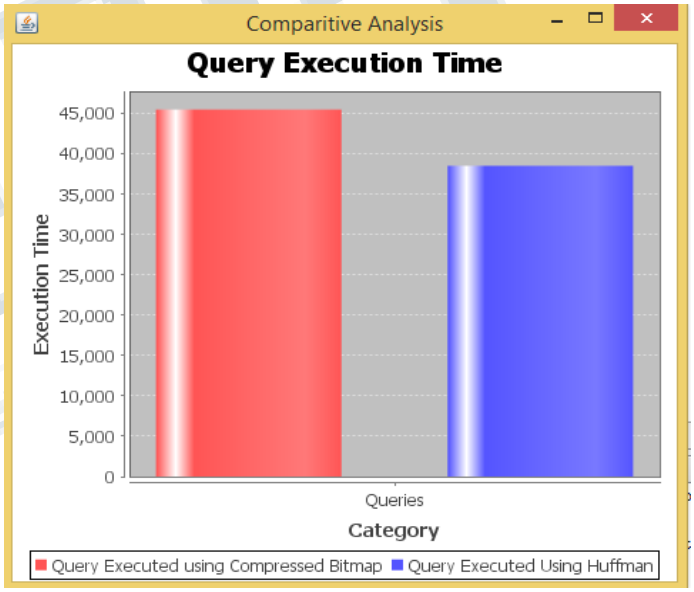

Figure 5. Result Analysis

Records are retrieved using all presented techniques. In Figure 3, time required to retrieve records using WAH Compression Technique is 45959 milliseconds. .In Figure 3, time required to retrieve the records using Variable Length Coding Technique is 38531 milliseconds. Result analysis is mentioned in figure 5. In proposed system, time required to retrieve records is much less than the existing system.

V. CONCLUSIONS

This paper presents an efficient algorithm for query processing and retrieving relevant records from datasets using compressed bitmap indices such as WAH compression Technique and VLC Huffman coding technique. Our algorithm demonstrates better performance over existing available schemes. We observed that bitmap index has benefits of providing bitmap to the records which results in disk access. Computation time can be reduced by conducting bitwise operations on Bitmap indexes. The problem of massive empty AND results, can be eliminated using algorithm with priority queues. We developed variable length encoding technique as optimization strategy to improve the performance of the system. The issue that we consider for future work, we will investigate in solutions when the data are of large size such that the bitmap of a single column does not fit in main memory.

REFERENCES

[1] Chengkai Li, Member,IEEE, Bin He, Ning Yan, Muhammad Assad Safiullah "Set Predicates in SQL: Enabling Set-Level Comparisons for Dynamically Formed Groups" IEEE Transactions on Knowledge and Data Engineering , Vol. 26, No. 2, FEBRYARY 2014.

[2] Chiranjeev D. Garhwani, Shreya S. Kandekar, Payal S. Chirde", A Review on Query Processing and Optimization in SQL with different Indexing Techniques",IJARCCE,Vol 5,Issue 1,pp no-329-332,2016.

[3] Bin He,Hui-l Hsiao, Member IEEE, Ziyang Liu ,Yu Huang,and Yi Chen,Member,IEEE, "Efficient Iceberg Query Evaluation Using Comressed Bitmap Index", IEEE Transactionson K1nowledge and Data Engineering , Vol. 24, No. 9, SEPTEMBER 2012.

[4] Jayant Rajurkar, T.Khan, "A System for Query Processing and Optimization in SQL for Set Predicates using Compressed Bitmap Index", IJSRD - International Journal for Scientific Research & Development, Vol. 3, Issue 02, 2015 | ISSN 2321-0613,pp no 798-801.

[5] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D.Ullman, "Computing Iceberg Queries Efficiently,"Proc. Int'l Conf.Very Large Data Bases (VLDB),pp. 299-310, 1998.

[6] J. Bae and S. Lee, "Partitioning Algorithms for the Computation of Average Iceberg Queries,"Proc. Second Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK),2000.

[7] K. Wu, E.J. Otoo, and A. Shoshani, "Optimizing Bitmap Indices with Efficient Compression, "ACM Trans. Database Systems,vol. 31, no. 1, pp. 1-38, 2006.

[8] P.E. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes,"Proc. ACM SIGMOD Int'l Conf. Management of Data,pp. 38-49, 1997.

[9] Jayant Rajurkar, T.K.Khan," Efficient Query Processing and Optimization in SQL using Compressed Bitmap Indexing for Set Predicates", *IEEE Sponsored 9th International Conference on Intelligent Systems and Control (ISCO)* Page No.619-623.DOI.10.1109/ISCO.2015.7282354.

[10] S. Melnik and H. Garcia-Molina, "Adaptive Algorithms for Set Containment Joins,"ACM Trans. Database Systems,vol. 28, no. 1, pp. 56-99, 2003.

[11] S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M.Tolton, and T. Vassilakis, "Dremel: Interactive Analysis of WebScale Data Sets,"Comm. ACM,vol. 54, pp. 114-123, June 2011.

[12] G. Antoshenkov, "Byte-Aligned Bitmap Compression,"Proc. Conf. Data Compression,p. 476, 1995.

[13] Jayant Rajurkar, Lalit dole, ," A Decision Support System for Predicting Student Performance", International Journal of Innovative Research in Computer and Communication Engineering(IJIRCCE). Vol. 2, Issue 12, December 2014, Pages- 7232-37.

[14] D. Chatziantoniou and K.A. Ross, "Querying Multiple Features of Groups in Relational Databases,"Proc. Int'l Conf. Very Large Databases (VLDB),pp. 295-306, 1996.

[15] D. Chatziantoniou and K.A. Ross, "Groupwise Processing of Relational Queries,"Proc. 23rd Int'l Conf. Very Large Databases (VLDB),pp. 476-485, 1997.

[16] D. Chatziantoniou and E. Tzortzakakis, "Asset Queries: A Declarative Alternative to Mapreduce,"ACM SIGMOD Record, vol. 38, no. 2, pp. 35-41, Oct. 2009.

[17] K. Wu, E. Otoo, and A. Shoshani, "An efficient compression scheme for bitmap indices" in ACM Transactions on Database Systems, 2004.

[18] K.Wu, E. J. Otoo, and A.Shoshani, "Compressing bitmap indexes for faster search operations" in Proceedings of the 2002 International Conference on Scientific and Statistical DatabaseManagement Conference (SSDBM'02), pages 99–108, 2002.

[19] Zainab Qays Abdulhadi, Zhang Zuping and Hamed Ibrahim Housien, **"**Bitmap Index as Effective Indexing for Low Cardinality Column in Data Warehouse" in International Journal of Computer Applications (0975 – 8887) Volume 68– No.24, April 2013.

[20] Sirirut Vanichayobon, Le Gruenwald,**"**Indexing Techniques for Data Warehouses Queries".

[21] https://en.wikipedia.org/wiki/B-tree.

[22] https://www.quora.com/What-is-a-B+-Tree.