A Review on Efficient Retrieval of Data from Large Datasets

Trupti Narnaware[1], Vibhuti Sharma[2], Jayant Rajurkar[3]

Department of Computer Engineering, MIET, Bhandara (MS), India[1, 2].

Assistant Professor, Department of Computer Engineering, MIET, Bhandara (MS), India[3]

Abstract - Here author should explain the brief summary about the research.In data warehousing and OLAP applications, huge amount of data are processed. So as to perform efficient retrieval of the data become highly inadequate which requires supporting set-level comparison semantics that means to compare a tuples of group with a number of values. We present here a comprehensive review of the state-of-the-art processing a tuples of group with multiple values and to optimize the queries. As the queries which are available now are complex, complex to write as well as challenging for database engine to optimize, which results in costly evaluation. Most of the available technique of the data 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 especially when data size is too large. Optimized query processing approach achieved by various studied algorithms shows very good performance to processing large datasets.

Keywords — Data Warehousing, OLAP, Bitmap index, Iceberg Query, Querying processing and optimization, Word-Aligned Hybrid(WAH), VLC Byte Aligned Bitmap Compression (BBC).

I. INTRODUCTION

Now a day's, need for querying a data in data warehouses and OLAP application with the semantics of set-level comparison is very high. If a company or institution needs candidates for the job with set of compulsory skills then a company or institution would search through their whole resume database. Skills of each candidate that is set of values are compared against the compulsory skills. Such sets are dynamically formed. By using currently available SQL syntax and semantics without proposed system such process of set level comparisons can be performed. If the set level comparisons performed using currently available SQL syntax, resulting query may be more and more complex; with the result it may take too much time to process the query than necessary. Such complex query becomes a difficult for the user to formulate, which results in too much costly evaluation.

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

In this paper, our aim is to process and retrieve the data using Bitmap indeces.Currently available GROUP BY clause can only and only do scalar value comparison by accompany HAVING clause. Aggregate functions COUNT, MIN, MAX, SUM and AVERAGE etc. produces single numeric value, which compared to another single aggregate value. We have presented Aggregate function based technique and compressed bitmap index based technique. Aggregate function based technique processes set predicates in the normal way as processing conventional aggregate function. Second technique is compressed bitmap index in which bitmap indices is created on each attributes. This technique is more efficient because it focuses on only those tuples which satisfies query condition and bitmaps of appropriate columns. Such index structure is applicable on many different types of attributes. This technique processes queries such as selections, joins, multi-attribute grouping etc [1].

II. RELATED WORK

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

Bin He et al.(2012) explained the properties of bitmap index and developed a very efficient and powerful bitmap index pruning strategy for processing queries.

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

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

Byte Aligned Bitmap Compression (BBC), Run-length encoding schemes accomplish compression when sequences of successive identical bits, and "runs", is presents. BBC [11] is an 8-bit hybrid RLE representation is in the practice of a literal or a fill. The MSB which are known as the flag bits marks the encoding type. That is, a byte 0xxxxxxx which will denote the least significant 7 bits is a literal representation for the genuine bit string. In distinction, 1xnnnnnn encodes a fill which compactly represents runs of consecutive x's. Here, x are the fill bit which encodes the value for the bits in the run, and the remaining 6 bits is use for length (in multiples of 7), e.g., 11001010 represented by the sequence of 70 1's.

Word Aligned Hybrid (WAH), compressed bitmap indexes are increasingly utilised for efficiently querying very large databases. The Word Aligned Hybrid (WAH) bitmap compression schemes are commonly recognized for the most efficient compression scheme in terms of CPU efficiency. WAH [16, 17], not like BBC that uses a 31 bit representation (32 bits including the flag bit). This representation offers several benefits over BBC—one being used for certain bitmaps, WAH can achieve significant speedup in query processing time duration when compared to BBC. These speedups 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 avoiding the overhead of the further extraction bytes within a word that is incurred by BCC. Thus, WAH not only compressed literals more effectively than BBC (using 4 less flag bits per 31 bits), but also it can also practice bitwise operations much quicker over literals by avoiding the overhead of byte abstraction or parsing and decoding to determine if the byte are indeed the literal.

In terms of compressing runs, however, Word aligned hybrid compression typically pales compared to BBC. This is often due to fact that WAH's fills are encode 2^{30}-1 multiples of 31 consecutive identical bits. In practice, runs for this size are unlikely, which implies that many of the fill bits are unused. On by the other hand, note this maximum number of consecutive bits that a BBC fill can represent is $(2^6-1)*7 = 441$. For large-scale and highly sparse databases, it is likely that a run can continue far beyond this threshold, which means there are still the cases where WAH will yield more efficient encodings for runs [11].

B-Tree is an self-balancing search trees. In most of the other self-balancing search trees like AVL and redly blackly trees, it is assuming that everything are in main memory. To understand use of B-Trees, we must think of large amounts of data that cannot fit in the memory.[20] When the number of keys is high, the data is read from disk in the forms of a block. Disk access time is very high compared to main memory access time. The main idea of using B-Trees is for reduced the numbers of disk accesses. Most of the tree operations (search, insert, delete, max, minuet) required 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 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 dislike Binary Search Tree. Binary Search Trees grow downward.

The B-Tree Index is popular in data warehouse applications for high cardinality column such as name since the space usage of the index is independent 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 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 keys and pointers to lower level nodes in the tree. These pointers can be thought of as being between each of the keys. To search for or insert an element into the tree, one load up the root node, find the adjacent keys that 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 a partial block, and a block can hold a large number of pointers.

B+ trees can also be used outside 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].

Variable Length Compression (VLC),Due to the use for the fixed bit-segment lengths to encode bit vectors, neither WAH nor BBC generates the optimal compression. Our scheme will provide an alternative to the user to encode a bitmap using precise encoding lengths to greater enhance compression, or to use encoding lengths that would allow for more rapidly querying on certain columns that may be queried often. Thus, VLC is a tuneable approach, which allows users to trade-off size and enactment [11].

A Huffman Coding is most sophisticated and efficient lossless data compression techniques. In Huffman Coding the typescripts in a data files are converted into 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 WORK

In proposed system we have presented Aggregate function based technique and bitmap index 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

Figure 1.An example of Bitmap index

Development of the bitmap compression method and encoding approaches further extends the applicability of bitmap indexing. Nowadays, this may be applied on all types of attributes like values of higher cardinality and categorical attributes numerical and text attributes. And it is very effectual for Online Analytical Process and warehouse query processing.

In the proposed work, first task is to conform whether the entire dataset is Bitmap indexed or not. Bitmap indexing is not applied to tuple having number of distinct term too much, as number of distinct term increases Bitmap table creation will be very complex. In order to

retrieve the data that contains tuples from such a dataset which is not Bitmap Indexed, will be retrieve with referenced to other data which is again needs to retrieve. The Request is send in the form of query to the database to retrieve the related data from the databases. The query is broken into the number of distinct query. The query is initially retrieve the relevant data from large dataset and check the relations with other attributes, then retrieve the related tuples. Then records with relevant attributes will retrieve from the datasets.

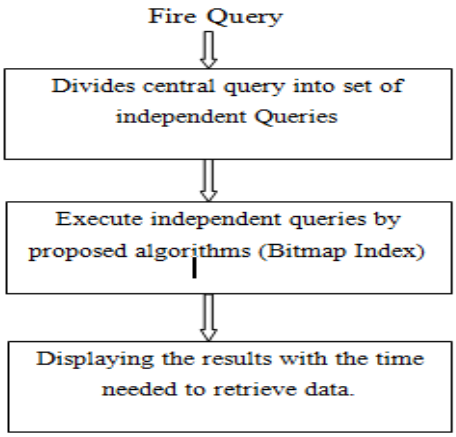

Figure 2. Flow of execution of proposed Algorithm.

IV. CONCLUSION

We have presented a comprehensive review on processing large data sets. Set predicative approach is used to process the large data by applying bitmap indexes, which allow selection of dynamically formed groups and set values. We have presented an approach, bitmap index based approach using to process large datasets. We observed that bitmap index has fallowing benefits:1)Saving disk access by avoiding tuple -scan on a table with more number of attributes, 2)Reducing computation time by conducting bitwise operations. We can further develop an optimization strategy to further improve the performance of the system.

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] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] G. Antoshenkov, “Byte-Aligned Bitmap Compression,”Proc. Conf. Data Compression,p. 476, 1995.

[14] 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.

[15] 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.

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

[18] 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.

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

[20] 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.

[21] Sirirut Vanichayobon, Le Gruenwald,**“**Indexing Techniques for Data Warehouses Queries”.

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