IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

A Review on Query Processing and Optimization in SQL with different Indexing Techniques

Chiranjeev D. Garhwani[1], Shreya S. Kandekar[2], Payal S. Chirde[3]

Dept. of Computer Science & Engg., Datta Meghe Institute of Engg., Technology & Research, Wardha (MS), India[1,2,3]

Abstract: In data warehousing and OLAP applications, large data are processed. To perform scalar-level predicates in SQL with large amount of 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. Currently available queries are complex, complex to write 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 especially when data size is too large. Optimized query processing approach achieved by various studied algorithms shows very good performance to processing set predicates.

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

I. INTRODUCTION

Now a day's, demand of querying the data in data warehouse and OLAP applications with the semantics of set-level comparison is very high. Suppose a company or institution seeking for candidates for the job with set of compulsory skills, company or institution may search their resume database. Skills of each candidate that is set of values are compared against the compulsory skills. Such 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 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 compressed 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]. For the purpose of compression Word-Aligned Hybrid (WAH) [5] technique is used. This technique now a day's can be applied on all types of attributes such as numeric attributes[6][7] high cardinality categorical attributes[6],text attributes[8] etc. This technique is efficient for data warehouse query processing and OLAP [9].

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

Copyright to IJARCCE

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

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.

1. 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.

BBC is compelling with query executed for the duration of the time is directly proportionalities is the rate of compressions. For example, suppose a database contains 77 rows and two bit vectors: v1 and v2. Assume that v1 contains the literal 0101010 followed by a run of 70 consecutive 1's. Let v2 contains the sequences of 700's following by the literal 0100000. In BBC format, v1 would be encoded as (00101010 11001010) and similarly, v2 = (10001010 00100000). Now envision a query which invokes v1 ^ v2. The query processor would read the first byte from both v1 and v2. By decoding the most significant bit, the query processor determines that it has read a 7-bit literal from v1 and a run of (10 * 7) = 70 0's from v2. Next, the literal from v1 is AND'ed with an fill of seven 0000000 from v2. Progressing further, the query processor reads and decodes the next byte from v1. It is important to note only seven 0's has been processed from the fill in v2. Thus, all that is required to simply decremented of the fill count from 10 to 9. This demonstrates why BBC fills must be a multiple of 7. The next byte of v1 is decoded as a run of 70 consecutive 1's. The next 9 AND operations can be carried out in one step by making the AND comparison once and reported its results for the same compressed form. The run-length count for v1 is updated to 1, and v2 to 0. Thus 63 = (9 * 7) bits have been compared without having to decode even once. After the 9th iteration, v2's fills are exhausted, prompted for the read of the next byte from v2. Finally, the remaining seven bit from both bins are AND'ed to complete the query. BBC's efficiently comes to presence of fills, which effectively allowed processor for amortizing the number of necessary memory accesses.

2. 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].

3. B- tree

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,

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

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].

4. B+ tree

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].

5. 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. To illustrate, recollected the rows reordered bitmaps produces long runs in the first several bit vectors, buts increasingly shorter run at the later vectors. Word Aligned Hybrid's 31-bit segment length (32 bits including the 1 flag bit) is idealistic for the first several bit vectors that potential for containing extremely long runs. But after this first few vectors, the rest may tend to has an average run-length smaller than 62 the straight run-length several that WAH can compress, there is a higher probability to have that many shorter runs to be represented as WAH literals, which squanders compression opportunities. Conversely, BBC's have an maximum fill codes, 1x1111111, can only represent a run of 63 * 7 = 441 x's. With this 7-bit fixed segmented length, BBC could not professionally represent long runs for the first several vectors. Any run that are longer than 441 would require another byte for the use. We posit that we can attain a balanced trade-off between these exemplifications by using unevenly-sized bit segment sizes. To this end, we propose a novel run-length compression scheme Variable Length Compression (VLC) which is capable of varying the segment lengths used for compression on a per bit vector basis. The flexibility of VLC permits us to bandage the preliminary bit vectors of a row reordered bitmap using a longer segment length, when used a smaller length on advanced bit vectors. While a more robust compression can be expected using VLC, a challenge is maintaining efficient query processing speed. VLC achieves greater compression in our experiments than both WAH and BBC at most cases, when the correct fragment interval is selected. 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 METHOD

In proposed system we have presented Aggregate function based technique and bitmap index based technique with Word-Aligned Hybrid (WAH) compression 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.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

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

Various compression schemes of the bitmap indexing has been developed. Word-Aligned Hybrid (WAH) and Byte-aligned Bitmap Code (BBC) is the two very important compression systems that can be applied to any column and be used in query processing without decompression. 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.

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 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. Below is the example of Variable Length Code.

	a	b	c	d	e	f
Freq in '000s	45	13	12	16	9	5
A fixed -length	000	001	010	011	100	101
A variable-length	0	101	100	111	1101	1101

Fig 2 .An example of VLC

The fixed length-code requires 300,000 bits to store the file. The variable length code uses only (45*1+13*3+12*3+16*3+9*4+5*4)*1000=224,000 bits.

IV. CONCLUSION

We have presented a comprehensive review on processing large data sets. Set predicates combined in a group, allow selection of dynamically formed groups and set values. We have presented an approach, compressed bitmap index based approach using variable length coding 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 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] 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.

[4] 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.

[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] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

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

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