Volume 4, Issue 9, September 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Review on Efficient Query processing for Set Predicates of Dynamically Formed Group

Jayant Rajurkar
PG Student
G. H. Raisoni College of Engineering,
Nagpur (MS), India

T. K. Khan
Assistant Professor
G. H. Raisoni College of Engineering,
Nagpur (MS), India

Abstract*— *In data warehousing and OLAP applications, scalar-level predicates in SQL become highly inadequate which needs to support set-level comparison semantics, i.e., comparing a group of tuples with several values. We present here a comprehensive review of the state-of-the-art processing a group of tuples with multiple values and to optimize the queries. Currently, complex SQL queries composed by scalar-level operations are difficult to write as well as challenging for database engine to optimize, which results in costly evaluation. Mostly available query processing algorithm does not take the advantage of the small-result-set property and heavily on the tuple-scan-based approach, which incurs intensive disk accesses and computations. This result in long processing time especially when data size is large. Optimized query achieved by various studied algorithms shows very good performance to processing set predicates.

Keywords*— *Bitmap index, Set predicates, Data Warehousing, OLAP, Querying processing and optimization.

I. INTRODUCTION

Knowledge discovery and business insight from operational data have been powerful weapons to take Competitive advantages in the modern business world [1]. In data warehousing and OLAP application, there is high demand of querying data with the semantics of set-level comparisons [2].Suppose a company maintains different databases within organization for specific operations. For instance , a company may search its resume database for job candidates with sets of mandatory skills. Here, skills of each candidate ,as a set of values are grouped ,are compared against the mandatory skills. Such sets are dynamically formed. In current scenario, a GROUP BY clause can only do scalar value comparisons by accompanying HAVING clause. For instance , aggregate functions COUNT /SUM/MAX/AVG produce single numeric value, which is compared to literal or another single aggregate value. Now a days, there is high demand of complex and dynamic set-level comparisons in databases, such a demand can be eliminate with the concept of set predicate. Below are some example queries with set predicates.

Example 1.To find those candidates with skills "PHP" and "Java" ,a query can be written as follows: After grouping , a dynamic set of values on attribute skill is formed for each unique id ,and groups whose corresponding SET(skill) contains both "PHP" and "Java" are returned:

SELECT id FROM Resume_Skill GROUP BY id HAVING SET(skill) CONTAIN {'PHP','Java'}

Example 2. The concept of Set predicates can be defined across multiple attributes .Consider an online advertisement example. Suppose the table schema is Site_Statistics(website, advertiser, CTR). A marketing strategist uses the following query to find websites that publish ads for ING with more than 2 percent click-through rate (CTR) and do not publish ads for HSBC yet:

SELECT website FROM Site_Statistics GROUP BY website HAVING SET(advertiser,CTR) CONTAIN {(ING,>0.02)} AND NOT (SET(advertiser) CONTAIN {'HSBC'})

In this example, the first set predicate involves two attributes and the second set predicate uses the negation of CONTAIN. Note that we use >0.02 to represent a range based condition CTR >0.02).

The semantics of set-level comparisons in lot of cases can be expressed using current SQL syntax without proposed extension. But, resulting queries would be more and more complex than necessary. One significance is that complex queries are difficult for users to formulate. Most importantly, such complex queries are difficult for DBMS to optimize, leading to unnecessarily costly evaluation[2]. The resulting query plans could involve multiple sub queries with grouping and set operations. The proposed syntax of set predicates enables direct expression of set-level comparisons in SQL, which makes query formulation simple as well as facilitates efficient support of such queries.

With the concept of threshold constraint, a special class of query such as iceberg query usually only returns a very small percentage of distinct groups as the output. Because of the small result set, iceberg queries can potentially be answered quickly even over a very large data set. Currently available database systems and/or approaches do not fully take advantage of feature of iceberg query. The relational database systems nowadays (e.g., DB2, MySQL, Oracle, Sybase, SQL Server, PostgreSQL, and column-oriented databases Vertica, LucidDB, MonetDB,) are all using general aggregation algorithms [4], [6], [5] to answer iceberg queries by first aggregating all tuples and then evaluating the

HAVING clause to select the iceberg result. For large data set, multipass aggregation algorithms are used when the full aggregate result cannot fit in memory. Most existing query optimization techniques for processing iceberg queries [9], [6] can be categorized as the tuple-scan-based approach, which needs at least one table scan to read data from disk. They focuses on the reducing number of passes when the data size is large[2]. None has effectively leveraged the property of iceberg queries for efficient processing. Such a tuple-scan-based scheme often takes a long time to answer iceberg queries, especially when the table is very large.

II. RELATED WORK

Set-valued attributes provide a more concise and natural way to model complex data concepts such as sets [2], [5]. Nowadays many DBMSs support the definition of attributes involving a set of values, for example, nested table in Oracle and SET data type in MySQL. For example, the "skill" attribute in Example 1 can be defined as a set data type. Set operations can be natively supported on such attributes. Although set-valued attributes together with set containment joins can support set-level comparisons, set predicates have several critical advantage of redesigning database storage foe special set data type set predicates require no change in data storage and representation , and thus can be incorporated into standard RDBMS[2].

In addition to this, Set predicate is also related to universal quantification and relational division [07], which are powerful for analyzing many-to-many relationships. An example universal quantification query is to find out the students that have taken all computer science courses needed to graduate. It is a special type of set predicates with CONTAIN operator over all the values of an attribute in a table, Courses. By contrast, the proposed set predicates allow sets to be dynamically formed through GROUP BY and support CONTAINED BY and EQUAL, in addition to CONTAIN[2].

Bitmap indices are known to be efficient, especially for read-mostly or append-only data, and are generally used in the data warehousing applications and column stores. Model 204 [1] was the first commercial product making extensive use of the bitmap index. Early bitmap indices are used to implement inverted files [08] In data warehouse applications, bitmap indices are. shown to perform better than tree-based index schemes, such as the variants of B-tree or R-tree [2]. Compressed bitmap indices are widely used in column-oriented databases, such as C-Store [9], which contribute to the performance gain of column databases over row-oriented databases.

Various compression schemes for bitmap index have been developed. Byte-aligned Bitmap Code (BBC) [10] ,Word-Aligned Hybrid (WAH) [11] are two important compression schemes that can be applied to any column and be used in query processing without decompression. The development of bitmap compression methods[11] and encoding strategies [12] further broaden the applicability of bitmap index. Nowadays, it can be applied on all numeric attributes, and text attributes And it is very efficient for OLAP and warehouse query processing. However, bitmap index is not effectively leveraged in existing works to process iceberg queries. In this paper, we develop query processing algorithms using bitmap .

III. EFFICIENT QUERY PROCESSING

Chengkai et al.(2014) presented two approaches to process set predicates, aggregate function based approach and Bitmap index based approach. Aggregate function-based approach. Processes set predicates in a way similar to processing conventional aggregate functions. Given a query with set predicates, instead of decomposing the query into multiple subqueries, Aggregate function-based approach approach only needs one pass of table scan attributes. It is efficient because it can focus on only the tuples from those groups that satisfy query conditions and only the for relevant columns. State-of-the-art bitmap compression methods [13] and encoding strategies [14] have made it affordable to build bitmap index on many attributes. This index structure is also applicable on many different types of attributes. The bitmap index based approach processes general queries (with selections, joins, and multiple set predicates and multi attribute grouping) by utilizing single-attribute indices. Hence, it does not require recomputed index for join results or combination of attributes.

3.1 QUERY PROCESSING USING COMPRESSEDBITMAP INDEXING

Bin He et al.(2012) exploited the property of bitmap index and developed a very effective bitmap pruning strategy for processing iceberg queries. Index-pruning-based approach eliminates the need of scanning and processing the entire data set (table) and thus speeds up the iceberg query processing significantly. This approach is much more efficient than existing algorithms commonly used in column oriented databases and row-oriented databases. Observing these attractive characteristics, the opportunities of computing iceberg queries efficiently using compressed bitmap index. A naive way for computing iceberg query using bitmap indices is to do pairwise bitwise-AND operations between bitmap vectors of all aggregate attributes. This is very inefficient because the product of the number of bitmap vectors in all aggregate attributes is large and a large portion of these operations are not necessary. Leveraging the antimonotone property of iceberg query, developed the dynamic pruning algorithm.

However, there is another challenge in the dynamic index-pruning-based approach—the problem of massive empty bitwise-AND results. When the number of unique values in an attribute is large, a large number of bitwise-AND operations produce empty results and the computation time dominates the query processing time. To overcome this challenge, developed an efficient vector alignment algorithm. The major challenge in developing such an algorithm is to effectively detect whether a bitwise-AND will generate empty result before doing the AND operation.

This sounds like a dilemma in the beginning, but after careful research, The vector alignment algorithm guarantees that any bitwise-AND operation will not generate empty result.

Elizabeth O'Neil et al .proposed FASTBIT and RIDBIT approaches. FastBit started research tool for studying how compression methods affect bitmap indexes, and has been shown since to be an efficient access method in a number of scientific applications [15]. It organizes data into tables (with rows and columns), where each table is vertically partitioned and different columns stored in separate files. Very large tables are also horizontal partitioned, each partition typically consisting of many millions of rows. A partition is organized as a directory, with a file containing the schema, followed by the data files for each column. This vertical data organization is similar to a number of contemporary database systems such as MonetDB, Sybase IQ, Kx systems and C-Store. FastBit implements a number of different bitmap indexes with various binning, encoding and compression strategies [15]. The index used in this study is the Word-aligned hybrid(WAH) compressed basic bitmap index. FastBit generates all bitmaps of an entire index for one partition in memory before writing the index file. This dictates that the entire index must fit in memory and imposes an upper bound on how many rows a horizontal partition can hold on a given computer system. Typically, a partition has no more than 100 million rows, so that a small number of bitmap indexes may be built in-memory at once.

Similarly RIDBit was developed as a pedagogical exercise for an advance database internals course, to illustrate how a bitmap indexing capability could be developed. The architecture of RIDBit is based on index design first developed for the Model 204 Database product from Computer Corporation of America [15]. The column values are represented as keyvalues in a B-tree index, and row-sets that follow each column value are represented either as RID-lists or bitmaps. Bitmaps more space-efficient than RID-lists when the bitmap is relatively dense, and bitmaps are usually more CPU-efficient as well. To create Bitmaps for the N rows of a table T = {r1, r2, …, rN}, we start with a 1-1 mapping m from rows of T to Z[M], the first M positive integers. In what follows we avoid frequent reference to the mapping m: when we speak of the *row number* of a row r of T, we will mean the value m(r).

3.2 VARIABLE-ALIGNED LENGTH (VAL) BITMAP INDEXING

Ryan Slechta et al. proposed the Variable-Aligned Length (VAL) bitmap index encoding framework, which generalizes the commonly used word-aligned compression techniques. VAL presented a variable aligned compression framework, which allows columns of a bitmap to be compressed using different encoding lengths. Such flexibility creates a tunable compression that balances the trade-off between query processing time and space. The variable format of VAL presents several unique opportunities for query optimization. Proposed approach of a dynamic encoding-length translation heuristic to process point queries. For range queries, propose several column orderings based on the bitmap's metadata: largest segment length first (lsf), column size (size), and weighted size (ws), empirical study over both real and synthetic data sets, dynamic translation selection scheme produces query execution times found that the weighted size column ordering significantly and consistently out-performs other ordering techniques.

IV. CONCLUSION

We have presented a comprehensive review on Set predicates to support set-level comparisons. Such predicates combined in a group, allow selection of dynamically formed groups and set values. We have presented two approaches, an aggregate function based approach and compressed bitmap index based approach to process set predicates. 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. The problem of massive empty AND results, can be eliminated using efficient vector alignment algorithm using priority queues. We further develop an optimization strategy to further improve the performance of the system.

REFERENCES

[1] 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 Transactions on Knowledge and Data Engineering, Vol. 24, No. 9, SEPTEMBER 2012 .

[2] 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.

[3] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-so-Foreign Language for Data Processing," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 1099-1110, 2008.

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

[5] 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.

[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] 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.

[8] K.-Y. Whang, B.T.V. Zanden, and H.M. Taylor, "A Linear-Time Probabilistic Counting Algorithm for Database Applications," ACM Trans. Database Systems, vol. 15, no. 2, pp. 208-229, 1990.

[9] Y. Ioannidis, "The History of Histograms (Abridged)," Proc. Int'l Conf. Very Large Databases (VLDB), 2003.

[10] N. Mamoulis, "Efficient Processing of Joins on Set-Valued Attributes," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 157-168, 2003.

[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] M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E.J. O'Neil, P.E. O'Neil, A.Rasin, Tran, and S.B. Zdonik, "C-Store: A Colum- OrientedDBMS," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp. 553-564, 2005.

[13] R. Elmasri and S. Navathe, Fundamentals of Database Systems. Addison-Wesley, 2011.

[14] D. Chatziantoniou, "Using Grouping Variables to Express Complex Decision Support Queries," Data Knowledge Eng., vol. 61, no. 1, pp. 114-136, 2007.

[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.

