COLUMN-STORE DATABASES: APPROACHES AND OPTIMIZATION TECHNIQUES

Tejaswini Apte[1], Dr. Maya Ingle[2] and Dr. A.K. Goyal[2]

[1]Symbiosis Institute of Computer Studies and Research
[2]Devi Ahilya Vishwavidyalaya, Indore

ABSTRACT

Column-Stores database stores data column-by-column. The need for Column-Stores database arose for the efficient query processing in read-intensive relational databases. Also, for read-intensive relational databases,extensive research has performed for efficient data storage and query processing. This paper gives an overview of storage and performance optimization techniques used in Column-Stores.

1. INTRODUCTION

Database system storage technology is mainly composed by Column-Stores (CS) and Row-Stores (RS) [1]. Currently, the majority of popular database products use RS, i.e. to store each record's attributes together, such as Microsoft SQL Server, Oracle and so on. A single disk writing action may bring all the columns of a record onto disk. RS system shows its high performance in the majority of database system applications like business data management. In contrast, RS system functions poorly for variety of analytical data applications, as the purpose of analytical operations is to gain new view point through data reading and to drive the production and implementation of plan.

Therefore, researchers proposed the idea of CS in read-optimized databases [1]. CS database is different from traditional RS database, because the data in table is stored in column and visited by column. However, the CS access method is not quite suitable for transactional environment (activities), where activities and data in rows are tightly integrated. During the execution on CS database, query on metadata is the most frequent kind of operation [1]. The metadata table is always presented in the form of tuples to the upper interface. The metadata access efficiency is most concerned problem for ad-hoc query operations for large database. To improve the efficiency of read and write operations of metadata queries, the most simple and effective way is, column to row operation of CS metadata table in buffer. The objective of this survey is to present an overview of CS features for read-intensive relational databases.

The organization of the paper is as follows: Section 2 elaborates CS approaches, its themes and aspects of general disciplines that help to improve performance for ad-hoc queries. Section 3 emphasizes on the areas of performance optimization in CS. Moreover, it attempts to clarify the skills useful for improving the performance in CS. We conclude in Section 4 with the summary.

DOI : 10.5121/ijdkp.2013.3508

2. COLUMN-STORE APPROACHES

There are different approaches proposed in the literature to build a CS namely; Vertical Partitioning, Index-Only Plans, and Fractured Mirror.

2.1 Vertical Partitioning

The most straightforward way to simulate CS in RS is through vertical partition. Tuple reconstruction is required to build a single row in fully vertically partitioned approach. For tuple reconstruction in vertical partition, an integer "position" column is been added to every partition, preferably primary key (Figure 1). For multi-attribute queries joins are performed on position attributes through rewriting the queries [2].

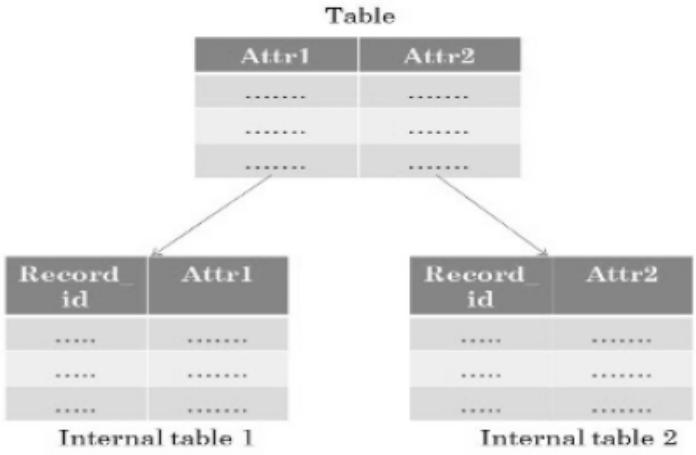

Figure 1: Vertical Partitioning

2.2 Index-Only Plans

The vertical partitioning approach wastes space by keeping the position attributes at each partition. To utilize the space efficiently, the second approach is index-only plans, in which each column of every table is indexed using unclustered B+Tree and base relations are stored using a standard, RS design. Index-only plan works through building and merging lists of (record-id, value) pairs that satisfy predicates on each table (Figure 2).

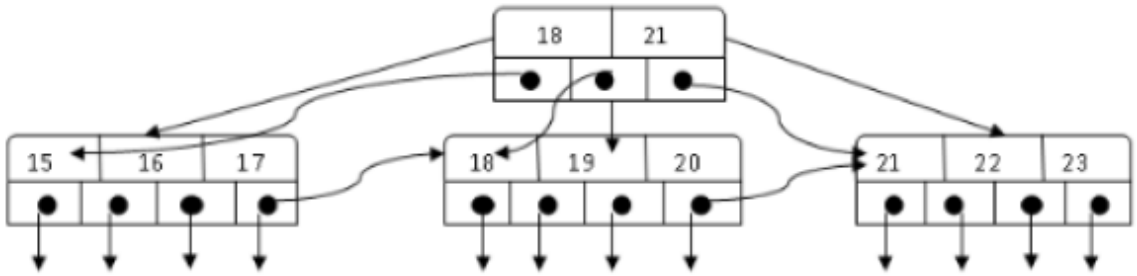

Figure 2:Index-Only Plans

Indices do not store duplicate values, and hence have less tuple overhead. For no predicate, query scanning through the index-only approach may be slower than scanning a heap file. Hence, a composite key indexes are created for an optimization through the index-only approach.

2.3 Fractured Mirrors Approach

This approach is driven by hybrid row/column approach. The design of fractured mirror includes RS for updates and the CS for reads, with a background processes migrating the data from RS to CS (Figure 3). Exploration of vertically partitioned strategy has been done in detail with the conclusion that tuple reconstruction is a significant problem, and pre-fetching of tuples from the secondary storage is essential to improve tuple reconstruction times [3].

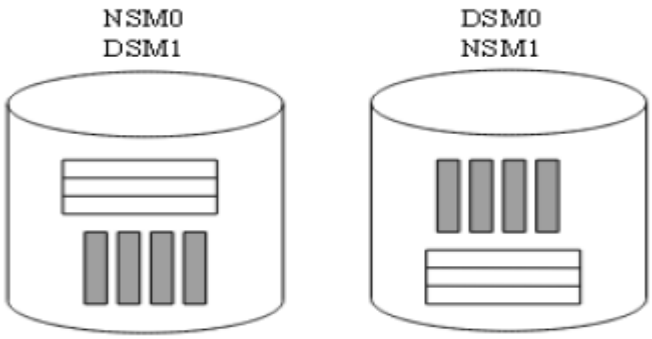

Figure 3 : Fractured Mirror Approach

3. OPTIMIZATION TECHNIQUES

Three common optimizations techniques are suggested in the literature for improving the performance in CS database systems.

3.1 Compression

The compression ratio through existing compression algorithms is more for CS, and has been shown to improve query performance [4], majorly through I/O cost. For a CS, if query execution can operate directly on compressed data, performance can further be improved, as decompression is avoided. Hu-Tucker algorithm is systematic and optimal logical compression algorithm for performing order preserving compression. The frequency of column values are considered to generate an optimal weighted binary tree. The dictionary is constituted from these frequent values [5].

In dictionary encoding logical compression algorithm, values are defined in schema and, in the source dataset these values are replaced by equivalent symbols, as the index value in the dictionary. To reduce the effect of decompression, indexes are used to scan the dictionary. An extensive research is being carried in dictionary based domain compression, which is extensively used in relational database. Dictionary encoding logical compression algorithm is frequently used for commercial database applications [6, 7].

Entropy encoding logical compression, including Huffman encoding [6], are considered heavy-weight algorithms, since decompressing variable length entropy encoded data is more processor intensive. These algorithms are modified to perform better in relational databases [7, 8]. Initially compression work mainly focused on disk space savings and I/O performance [7, 10, 11]. In addition literature shows that, with lower decompression cost; compression can also lead to better CPU performance [12, 13]. There exists important aspect related to the cost calculation of query optimizer in recent database literature.

3.2 Materialization Strategies

For read-intensive relational databases, vertical partitioning plays vital role for performance improvement. Recently several read-intensive relational databases have adopted the idea of fully vertically partition [1, 14]. Research on CS has shown that for certain read-mostly workloads, this approach can provide substantial performance benefits over traditional RS database systems. CS are essentially a modification only to the physical view of a database and at the logical and view level, CS looks identical to a RS.

Separate columns must ultimately be stitched into tuples. Determining the stitching schedule in a query plan is critical. Lessons from RS suggest a natural tuple construction policy i.e. for each column access, column is being added to an intermediate tuple representation for later requirement. Adding columns to the intermediate results as early as possible in the literature is known as early materialization. Early materialization approach in CS is more CPU bound. Late materialization approach is more CPU efficient, as less intermediate tuples to be stitched. However, sometime rescanning of the base columns is required to form tuples, which can be slow [15, 16].

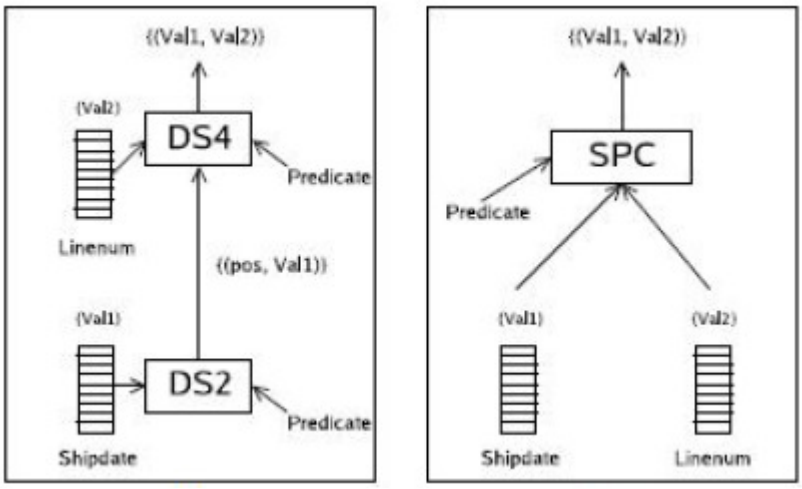

Figure 4 : Early Materialization Query Plan

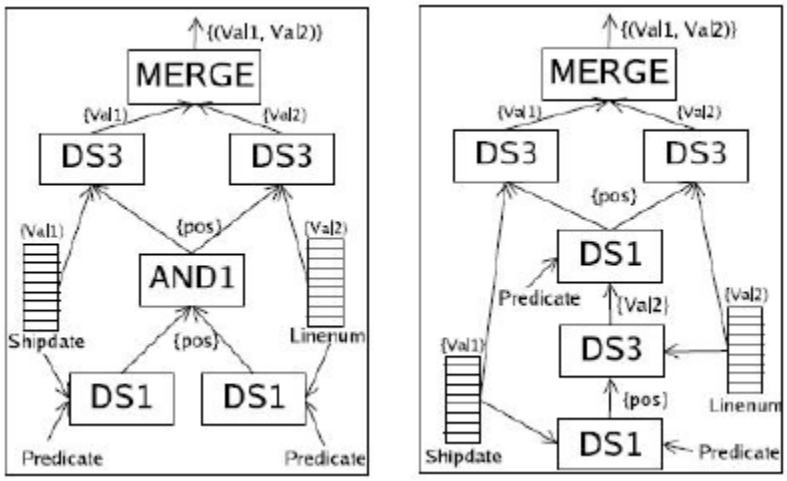

Figure 5: Late Materialization Query Plan

To overcome disadvantages, an invisible join can be used in CS for foreign-key/primary-key joins by minimizing the values that need to be extracted in arbitrary order. The joins are rewritten for predicates of foreign key columns. Predicate evaluation is carried through hash lookup. Only after evaluation of predicates the appropriate tuples are extracted from the relevant dimensions [17].

The successful CS systems C-Store and Vertica, exploit projection to support tuple reconstruction. They divide logically a relation into sub-relations, called projections. Each attribute may appear in more than one projection. The attributes in the same projection are sorted on one or more sort key(s) and are stored respectively on the sorted order [18]. Projection makes multiple attributes referenced by a query to be stored in the same order, but not all the attributes be the part of the projection and hence pays tuple reconstruction time penalty. In the past, the series of CS databases C-Store and MonetDB have attracted researchers in CS area due to their good performance [18].

C-Store series exploit projections to organize logically the attributes of base tables. Each projection may contain multiple attributes from a base table. An attribute may appear in more than one projection and stored in different sorted order. The objective of projections is to improve the tuple reconstruction time. Though the issue of partitioning strategies and techniques has not been addressed well, a good solution is to cluster attributes into sub-relations based on the usage patterns. The Bond Energy Algorithm (BEA) was developed to cluster the attributes of the relation based on compatibility [19].

MonetDB proposed self-organization tuple reconstruction strategy in a main memory structure called cracker map [14]. Split-up and combination of the pieces of the existing relevant cracker maps make the current or subsequent query select qualified result set faster. But for the large databases, cracker map pays performance penalty due to high maintenance cost [14]. Therefore, the cracker map is only adapted to the main memory database systems such as MonetDB.

3.3 Block Searching

Block Searching in CS is greatly influenced by the address lookup process. Block search process has been carried through hashing algorithms [4, 20, 1]. Hashing algorithms have been used widely to avoid block conflicts in multiprocessor systems. To compute the block number, integer arithmetic is widely used in linear skewing functions. Burroughs Scientific Processor introduced fragmentation to minimize the block search time. For stride patterns, block conflicts may be reduce by XOR-based hash functions. XOR-based functions design mainly focused on conflict-free hash function for various patterns [22]. Bob Jenkins' hash produces uniformly distributed values. However, the handling of partial final block increases the complexity of Bob Jenkins' hash [21].

4. CONCLUSION

In this paper, we presented various CS approaches to optimize performance of read-intensive relational database systems. CS enables read-intensive relational database engineers to broaden the areas of storage architecture and design, by exploration of entities, attributes and frequency of data values. Moreover, all disciplines require knowledge that constitutes data structures, data behaviour, and relational database design.

REFERENCES

[1] S. Harizopoulos, V.Liang, D.J.Abadi, and S.Madden, “Performance tradeoffs in read-optimized databases,” Proc. the 32th international conference on very large data bases, VLDB Endowment, 2006, pp. 487-498.

[2] S. Navathe, and M. Ra. “Vertical partitioning for database design: a graphical algorithm”. ACM SIGMOD, Portland, June 1989:440-450

[3] Ramamurthy R., Dewitt D.J., and Su Q. A Case for Fractured Mirrors. Proceedings of VLDB 2003, 12: 89–101

[4] Daniel J. Abadi, Samuel R. Madden, and Miguel Ferreira. Integrating compression and execution in column oriented database systems. In SIGMOD, pages 671–682, Chicago, IL, USA, 2006.

[5] http://monetdb.cwi.nl/, 2008.

[6] J. Goldstein, R. Ramakrishnan, and U. Shaft, “Compressing relations and indexes,” Proc. ICDE 1998, IEEE Computer Society, 1998, pp. 370-379.

[7] G. V. Cormack, “Data compression on a database system,” Commun.ACM, 28(12):1336-1342, 1985.

[8] B. R. Iyer, and D. Wilhite, “Data compression support in databases,” Proc. VLDB 1994, Morgan Kaufmann, 1994, pp. 695-704.

[9] H. Lekatsas and W. Wolf. Samc: A code compression algorithm for embedded processors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18(12):1689–1701, December 1999.

[10] G. Ray, J. R. Haritsa, and S. Seshadri, "Database compression: A performance enhancement tool," Proc. COMAD 1995, 1995.

[11] L. Holloway, V. Ramon, G. Swart, and D. J. Dewitt, "How to barter bits for chronons: compression and bandwidth trade offs for database scans," Proc. SIGMOD 2007, ACM, 2007, pp. 389-400.

[12] D. Huffman, "A method for the construction of minimum redundancy codes," Proc. I.R.E, 40(9), pp. 1098- 1101, September 1952.

[13] C. Lefurgy, P. Bird, I. Chen, and T. Mudge. Improving code density using compression techniques. In Proceedings of International Symposium on Microarchitecture (MICRO), pages 194–203, 1997.

[14] D. 1 Abadi, D. S. Myers, D. 1 DeWitt, and S. Madden, "Materialization strategies in a column-oriented DBMS," Proc. the 23th international conference on data engineering, ICDE, 2007, pp.466-475

[15] S. Idreos, M.L.Kersten, and S.Manegold, "Self organizing tuple reconstruction in column-stores," Proc. the 35th SIGMOD international conference on management of data, ACMPress, 2009, pp. 297-308

[16] H. I. Abdalla. "Vertical partitioning impact on performance and manageability of distributed database systems (A Comparative study of some vertical partitioning algorithms)". Proc. the 18th national computer conference 2006, Saudi Computer Society.

[17] Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In VLDB '01, pages 169–180, San Francisco, CA, USA, 2001.

[18] R. Agrawal and R. Srikanr."Fast algorithms for mining association rules,". Proc. the 20th international conference on very large data bases,Santiago,Chile, 1994.

[19] R. Raghavan and J.P. Hayes, "On Randomly Interleaved Memories,"SC90: Proc. Supercomputing '90, pp. 49-58, Nov. 1990.

[20] Gennady Antoshenkov, David B. Lomet, and James Murray. Order preserving compression. In ICDE '96, pages 655–663. IEEE Computer Society, 1996.

[21] Jenkins Bob "A hash function for hash Table lookup".

[22] J.M. Frailong, W. Jalby, and J. Lenfant, "XOR-Schemes: A Flexible Data Organization in Parallel Memories," Proc. 1985 Int'l Conf.Parallel Processing, pp. 276-283, Aug. 1985.