International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Column Oriented Database: Implementation and Performance Analysis

Vibha Shukla[1], Dr. Rajdev Tiwari[2]

[1, 2]Noida Institute of Engineering and Technology, UPTU, India

Abstract: ***The volume of data in an organization is growing rapidly. So does the number of users who need to access and analyze this data. IT systems are used more and more intensive, in order to answer more numerous and complex demands needed to make critical business decisions. Data analysis and business reporting need more and more resources. Therefore, better, faster and more effective alternatives have to be found. Column oriented database systems have an important demand in the past few years. These databases are more suitable for data warehousing system to get analysis done faster as data is stored in columnar form. This paper provides a brief review of several papers, articles to know about the background of column oriented database. In this paper, we study the method for implementing column-store on top of Row-store in PostgreSql along with successful design and implementation. Performance results of Column-Store are presented, and compared with that of Row-store results with the help of TPC-H benchmark.***

Keywords: Column-oriented database, data warehouse, row-oriented database

1. Introduction

Row and column-oriented storage structures serve different purposes. It is more beneficial to use row-oriented storage structure if there are mostly transaction queries performed on a database. Transaction (OLTP-style) queries imply a set of reads and writes to a few rows at a time. . However, column-oriented storage structure is more beneficial if there are mostly analytical queries performed on a database. Faced with massive data sets, a growing user population, and performance driven service level agreements, organizations everywhere are under extreme pressure to deliver analyses faster and to more people than ever before. That means Businesses need faster data warehouse performance to support rapid business decisions, added applications, and better system utilization. And as data volumes continue to increase driven by everything from longer detailed histories to the need to accommodate big data companies require a solution that allows their data warehouse to run more applications and to be more responsive to changing business environments.

The row by row approach is write optimized. The three most popular commercial database systems (e.g., Oracle, IBM DB2, Microsoft SQL Server) choose the row by row storage layout. The column by column approach is read optimized. Suppose any table that have 5 columns and that table contain 10 million records. (e.g. customer (custid, custname, phone, email, sex)). But we frequently access only two records (e.g. custid, custname), so there is no need to read all the data from a particular table. Instead of reading all data we read only two columns.

2. Literature Review

Column-oriented databases allow data to be stored column-by-column rather than row-by-row. Research performed by Abadi, Madden, and Hachem (2008) proved that column-stores were faster than row-stores when reading large datasets optimized for analysis. This conclusion was based on four main advantages that revealed themselves in experiments: late materialization, block iteration, Compression, and invisible joins. All four of these advantages are essentially products of a column-store's ability to scan columns separately and track whole records by their identical position in each column.

Research done by Holloway and DeWitt (2008) focused on situations where the column-store's advantages in query speed and the use of compression did not materialize and where row-stores could match or even exceed their performance. One of the main reasons that a traditional row-store cannot match a column-store's performance when reading large datasets is the row-store's focus on easy updatability. Row-stores that sacrifice this advantage and reorganize themselves for "read-optimized" performance can achieve data compression ratios comparable to column-stores, and can exceed their query performance under certain conditions.

Loshin (2009) concurred, and also added that the pre-calculated aggregates and specialized views needed in row-store data warehouses were also unnecessary when utilizing a column-store because of the latter's high calculation speed. He went on to note that the removal of these elements, as well as a column-store's general lack of indexes, can streamline a database schema and greatly reduce both storage space and the complexity of extract, transfer and load processes (ETL).

Finally, he stated that the smaller amount of storage needed by a column-store can also be used more efficiently because the flexibility allowed by block iteration and invisible joins can make it easier to allocate disk resources, especially in shared-disk systems.

2.1 Advantages of Column-Stores

Column-oriented organizations are more efficient when an aggregate needs to be computed over many rows but only for a notably smaller subset of all columns of data, because reading that smaller subset of data can be faster than reading all data.

1) High performance on aggregation queries (like COUNT, SUM, AVG, MIN, MAX).
2) Highly efficient data compression and/or partitioning.
3) True scalability and fast data loading for Big Data.
4) Advantage of column oriented databases over row oriented databases is in the efficiency of hard- disk access.

Due to their aggregation capabilities which compute large numbers of similar data items, column oriented databases offer key advantages for certain types of systems, including: Data Warehouses and Business Intelligence, Customer Relationship Management (CRM),Library Card Catalogs, Ad hoc query systems.

2.2 Disadvantages of column-stores

2.2.1 Increased Disk Seek Time

Disk seeks between each block read might be needed as multiple columns are read in parallel. However, if large disk pre-fetches are used, this cost can be kept small.

2.2.2 Increased Cost of Inserts

Column-stores perform poorly for insert queries since multiple distinct locations on disk have to be updated for each inserted tuple (one for each attribute). This cost can be alleviated if inserts are done in bulk.

3. Implementation

Now the goal is to design column oriented databases and to propose new ideas for performance optimization. Our approach of implementing column oriented database is to vertically partition a traditional row oriented database. Tables in the row store are broken up into multiple two column tables consisting of (table key, attribute) pairs. There is one two column tables for each attribute in the original table. When a query is issued, only those thin attribute-tables relevant for a particular query need to be accessed-the other tables can be ignored. These tables are joined on table key to create projection of original table containing only those columns necessary to answer a query, and then execution proceeds as normal. The smaller the percentage the columns from table that need to be accessed to answer a query the better the relative performance with a row store will be.

This approach requires no modification to the database and can be implemented in all current database system. If it performs well, then it would be the preferable solution for implementing column stores. Of course this approach does require changes at the application level since the logical schema must be modified to implement this approach so this approach is to use a row-store to simulate a column-store. For performance analysis of row oriented database vs column oriented database there is a need of large row-oriented database. Using this large row-oriented database column-oriented database can be derived by vertical partitioning. Analysis of performance will be based on execution time of sql queries on the row oriented database and column oriented respectively.

To accomplish this, we Our benchmarking system TPC-H is the gold standard, and we use a simplified version of this benchmark with 2 Gbytes of memory and 750 Gbytes of disk, which our current engine is capable of running. Specifically, we implement the lineitem, and lineitemv tables as follows:

The standard data for the above table schema for TPC-H scale is 60,000,000 recods (1.8GB), and was generated by the data generator available from the TPC website. By vertical partitioning And creating the Trigger Function Linerec() We have taken lineitemv table in column oriented fashion with different numbers of columns and a separate file has been made.

SQL query that is being used for analysis purpose in column oriented database is for eight columns accessed is SELECT l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_receiptdate, l_shipinstruct, l_shipmode, FROM lineitemv; In the same way, we accessed different numbers of columns and check the performance of row and column oriented database at certain conditions on postgresql. The following table indicates the performance that we observed. All measurements are in seconds and are taken on a dedicated machine.

Table 1: Experimental results for simple select query

Sno	Approah	Table	No. of Column	Execution Time (in sec)
1	Row Column	LineItem LineItemv	16 6	45.89 13.45
2	Row Column	LineItem LineItemv	16 8	45.89 16.36
3	Row Column	LineItem LineItemv	16 10	45.89 20.34
4	Row Column	LineItem LineItemv	16 14	45.89 39.3

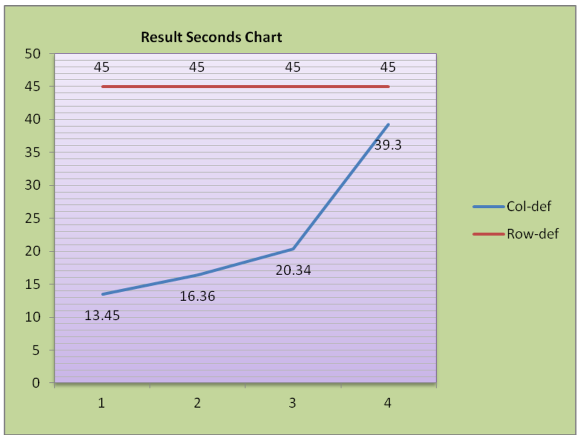

Figure: Comparison of Column-Store vs. Row-Store

A comparative analysis can be done that which database software will perform better after vertical partitioning for column oriented database. In this, we can see for the same query as increasing the number of columns time vary accordingly. For That analysis will certainly help in choosing row-oriented database software for column oriented database development. The same sql query that was used in query performance analysis will be used for comparative performance analysis.

4. Future Work

Vertical partitioning is a good approach for column oriented database design but this approach doesn’t make logical data independence in the design. There is a need of designing an automatic query rewriter that automatically converts queries over the initial schema to query over the vertically partitioned schema. This approach also introduces extra redundancy in the database. So instead of using primary key or serial no indexing can be used.

5. Conclusion

Column-oriented databases provide faster answers, because they read only the columns requested by users’ queries, since row-oriented databases must read all rows and columns in a table. For applications that write and update many data (OLTP systems), a row-oriented approach is a proper solution. In contrast, an OLAP system, mainly based on adhoc queries performed against large volumes of data, has to be read optimized.

Vertical partitioning approach to build a column-store requires slight modifications in the DBMS. This modification in the DBMS will certainly result is significant performance gains for large databases. It will certainly be useful for data warehouses where the analysis is naturally a read oriented endeavor. Unlike row oriented databases write optimized nature column oriented databases will be read optimized. This approach is not useful at all cases but it will certainly improve performance on some cases.

References

[1] A Review of Column-oriented Data stores by Zach Pratt at attackofzach.com
[2] Column-Oriented Databases, an Alternative for Analytical Environment by Gheorghe MATEI at dbjournal.ro
[3] Daniel J. Abadi “Query Execution in Column-Oriented Database Systems” Massachusetts Institute of Technology, February 2008
[4] D. J. Abadi, S. R. Madden, and M. C. Ferreira. In SIGMOD, pages 671-682, 2006, Integrating Compression and Execution in Column-Oriented Database Systems
[5] D. J. Abadi, S. Madden, N. Hachem, “Column-stores vs. row-stores: how different are they really?” in SIGMOD Conference 2008, pp. 967-980.
[6] Column-oriented DBMS, Wikipedia, the free encyclopedia.Available: http://en.wikipedia.org/wiki/Column-oriented_DBMS.
[7] Infobright, Analytic Applications With PHP and a Columnar Database(2010), 403-4Colborne St Toronto, Ontario M5E 1P8 Canada.
[8] Column-Oriented Databases, an Alternative for Analytical Environment by Gheorghe MATEI at dbjournal.ro