Performance Analysis of Column Oriented Database versus Row Oriented Database

Amit Kumar Dwivedi
Deptt of IT
IPEC, Ghaziabad

C.S.Lamba
Deptt of CS
RIET, Jaipur

Shweta Shukla
Deptt of CS
RIET, Jaipur

ABSTRACT

There are two obvious methods to map a two-dimension relational database table onto a one-dimensional storage interface: store the table row-by-row, or store the table column-by-column. Traditionally, database system implementations and research have focused on the row-by row data layout, since it performs best on the most common application for database systems: business transactional data processing. However, there are a set of emerging applications for database systems for which the row-by-row layout performs poorly. These applications are more analytical in nature, whose goal is to read through the data to gain new insight and use it to drive decision making and planning.

In this paper, we study the poor performance of row-by-row data layout for these emerging applications, and evaluate the column-by-column data layout opportunity as a solution to this problem. The solution will be analyzed and represented by graph. At the end of the paper we will see the comparative performance of Oracle 10g and MSSQLServer database.

Keywords

Databases, Database Systems, Row Store, Column Store, Performance Tuning

1. INTRODUCTION

The world of relational database is a two dimensional world. Here data is stored in form of tabular data structures where rows correspond to distinct real-world entities, and columns are attributes of those entities. For example, a college may store information of its students in a database table where each row stores information about a different student and each column stores a particular student attribute (name, e-mail, year, branch, etc.)

By this approach we can say that database is a two dimensional concept. But this property exists only at conceptual level. At a physical level database tables need to be mapped onto one dimensional structure before being stored. This is because of computer storage media (e.g. magnetic disk or RAM).

1.1 Rows vs Columns

There are two ways to map database tables onto one dimensional interface: store the table row by row or store the table column by column. The row by row approach keeps information about an entity together. In the student example above, it will store all information about the first student and then all information about the second student and so on. The column by column approach keeps all attribute information together: all of the student name will be stored consecutively and then all of the student email, etc.

Both approaches are well designed and typically a choice is made based on performance expectation if the expected workload is based on entity (e.g., find a student, add a student etc) then row by row approach is preferable because all information regarding a particular student is stored together but if the expected workload tends to read per query only few attributed from many records (e.g., a query that finds most common email address domain), then column by column storage is preferable since irrelevant attributes for a particular query do not have to be access. The row by row approach is writing optimized. The three most popular commercial database systems (e.g., Oracle, IBM DB2, Microsoft SQL Server) choose the row by row storage layout. The design was optimized for the most common application at a time: business transactional data processing. The goal of these applications was to automate mission-critical business tasks. For example, a bank might want to use a database to store information about its branches and its customers and its accounts. Typical use of this database might be to find the balance of a particular customer's account or to transfer 100 rupee from customer A to customer B in one single atomic transaction .These queries commonly access data on the granularity an entity(find a customer, or an account, or branch information, add a new customer, account, or branch). Given the workload, the row-by -row storage layout was chosen

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. RELATED WORK

It is generally accepted that data warehouses and OLAP workloads are excellent applications for column-stores. Column-stores do not see a performance degradation when storing extremely wide tables. Column-stores may be good storage layers for Semantic Web data, XML data, and data with GEM-style schemas [1].

2.1 Advantages of column store

2.1.1 Improved bandwidth utilization: In a column-store only those attribute that is accessed by a query needs to be read- off disk (or from memory into cache). In a row store surrounding attributes also need to read since the attribute is generally smaller than the smallest granularity in which data can be accessed [2].

2.1.2 Improved data compression: Storing data from the same attribute domain increases locality and thus data compression ratio (especially if the attribute is sorted). Bandwidth requirements are further reduced when transferring compressed data [3].

2.1.3 Improved code pipelining: Attribute data can be iterated through directly without indirection through a tuple interface. This results in high IPC (instructions per cycle) efficiency, and code that can take advantage of the super-scalar properties of modern CPUs [4, 5].

2.1.4 Improved cache locality: A cache line also tends to be larger than a tuple attribute, so cache lines may contain irrelevant surrounding attributes in a row-store. This wastes space in the cache and reduces hit rates [6].

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.

2.2.3 Increased tuple reconstruction costs: In order for column-stores to offer a standards-compliant relational database interface (e.g., ODBC, JDBC, etc.), they must at some point in a query plan stitch values from multiple columns together into a row-store style tuple to be output from the database. Although this can be done in memory, the CPU cost of this operation can be significant. In many cases, reconstruction costs can be kept to a minimum by delaying construction to the end of the query plan [7]

3. IMPLEMENTATION

Now the goal is to design column oriented databases and to propose new ideas for performance optimization. One 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.

In a fully vertically partitioned approach, some mechanism is needed to connect fields from the same row to together (column stores typically matchup records implicitly by storing columns in the same order, but such optimization are not available in a row store). To accomplish this, the simplest approach is to add an integer "position" column to every table- this is often preferable to use the primary key because primary keys can be large and are sometimes composite. This approach creates one physical table for each column in the logical schema. By the example given below the conversion of a row by row database to column oriented database can be shown.

Table1. Student

SNO	STUNAME	PHONE	EMAIL	DOB
001	AMIT	9811666473	ami@gmail.com	14-JAN-1988
002	PRABHAT	9811456689	pra@gmail.com	27-MAR-1989
003	VIKAS	9874566884	vik@gmail.com	05-AUG-1990

Table2. Student table after vertical partitioning

SNO	EMAIL
001	ami@gmail.com
002	pra@gmail.com
003	vik@gmail.com

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. In this paper Oracle 10g and MS-SQLServer2008 R2 is taken as database software.

There are two table in the database name EMPPERSONAL (SNO, EMPID PHONE, ADDRESS, PIN) and

EMPPROF(SNO,EMPID,SALARY). Initially both tables contains 2 million records each. By Vertical partitioning on the tables(EMPPERSONAL, EMPPROF) three tables have been derived from both table and a separate database has been made. The three tables are EMPX(SNO,PHONE), EMPY(SNO,ADDRESS) and EMPZ(SNO,SALARY) respectively. Now queries will execute on this database and the performance will analyzed on the basis of query execution time(in sec).

3.1 Analysis of Performance

Performance will be analyzed on software Oracle 10g and MSSQLSERVER 2008 R2. The tool that has been used for front end analysis is MATLAB 7.6.0(R2008 a).

3.1.1 On Oracle 10g: Sql query that is being used for analysis purpose on row oriented database is "select empprof.salary from empprof,emppersonal where emppersonal.address='COTRJX' and empprof.sno =emppersonal.sno'"

Sql query that is being used for analysis purpose in column oriented database is "select empz.salary from empz,empy where empy.address='COTRJX' and empz.sno=empy.sno"

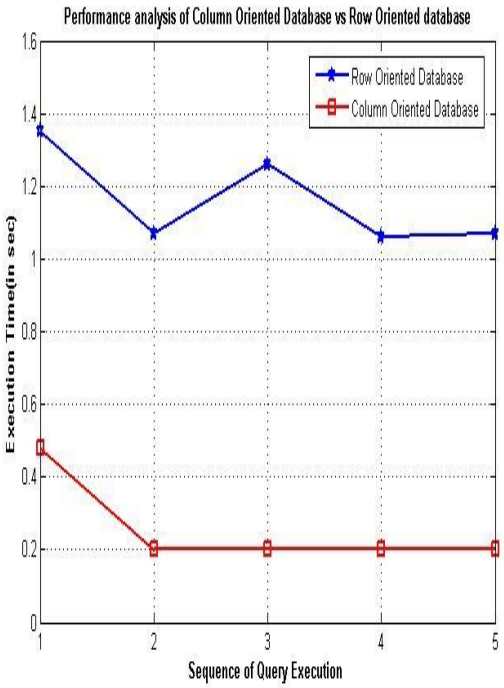

Fig1: Query Performance on Oracle 10 g database

By Fig1 it is clear that Column Oriented database performs better than Row-Oriented database at certain conditions on Oracle 10g.

3.1.2 On SQL SERVER 2008 R2: Sql query that is being used for analysis purpose on row oriented database is "select empprof.salary from empprof,emppersonal where emppersonal.address='COTRJX' and empprof.sno =emppersonal.sno'"

Sql query that is being used for analysis purpose in column oriented database is "select empz.salary from empz,empy where empy.address='COTRJX' and empz.sno=empy.sno"

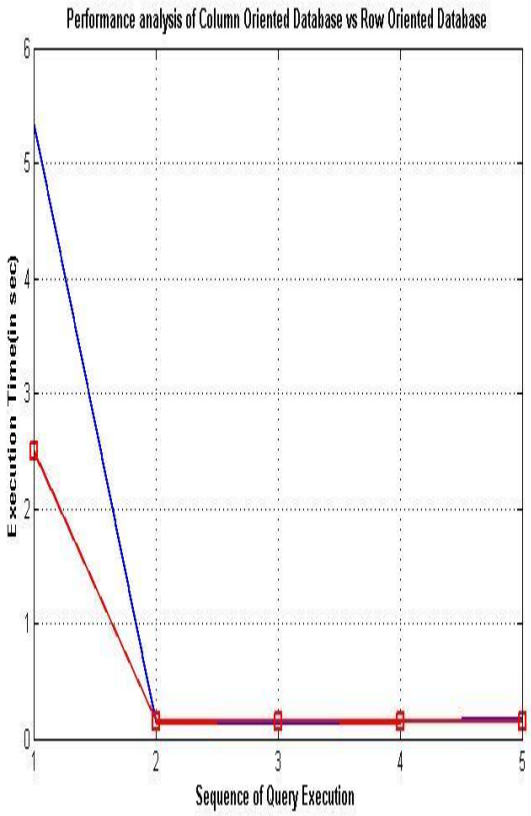

Fig2: Query Performance on MSSQLSERVER2008R2 database

By Fig2 it is clear that Column Oriented database performs better than Row-Oriented database at certain conditions on MSSQLSERVER 2008R2.

3.1.3 Comparative performance analysis of Column Oriented database on Oracle 10gand MSSQLServer 2008R2:

A comparative analysis can be done that which database software will perform better after vertical partitioning for column oriented database. 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.

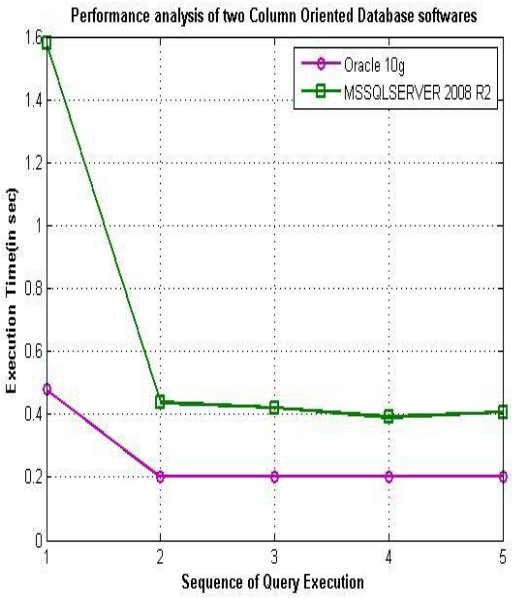

Fig3: Comparative query Performance on Oracle 10g and MSSQLSERVER2008R2 database

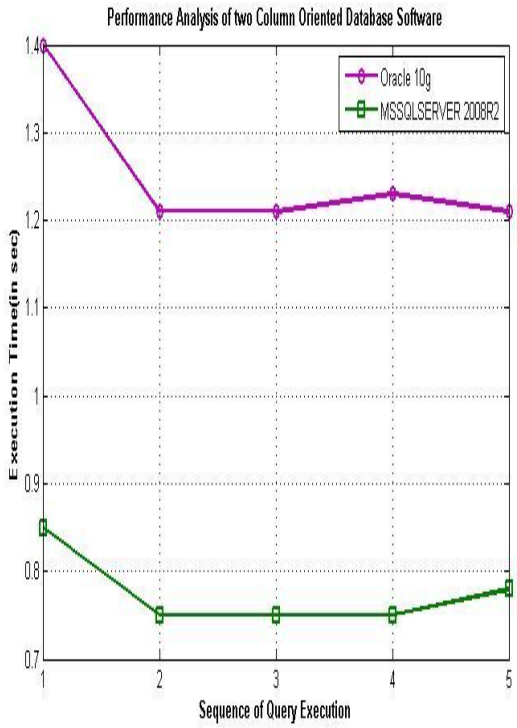

Fig4: Comparative query Performance on Oracle 10g and MSSQLSERVER2008R2 database

The comparative performance of both the databases can be understood by the above diagram and on the basis of Fig 3 and Fig 4 it is clear that in some cases Oracle 10g performs better than MSSQLSERVER 2008 R2 for Column Orientation and in some cases MSSQLSERVER 2008 R2 performs better than Oracle 10g. So we any database software either Oracle 10g or MSSQLSERVER 2008 R2 can be chosen for Column-Orientation.

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 degining an automatic query rewriter that automatically converts queries over the initial achema to query over the vertically partitioned schema.This approach alsso introduces extra redundancy in the database. So instead of using primary key or serial no indexing can be used..

5. CONCLUSION

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 endeavour. 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.

6. REFERENCES

[1] Daniel J. Abadi., In CIDR, Asilomar, CA, USA, 2007 Column stores for wide and sparse data.

[2] S. Khosha_an, G. Copeland, T. Jagodis, H. Boral, and P. Valduriez. In ICDE, pages 636-643,1987, A query processing strategy for the decomposed storage model.

[3] 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.

[4] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100:, In CIDR, 2005, Hyper-pipelining query execution.

[5] P. A. Boncz and M. L. Kersten. MIL primitives for querying a fragmented world.

[6] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, pages 169-180, 2001,Weaving relations for cache performance. In VLDB.

[7] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, In ICDE, 2007,Materialization strategies in a column-oriented dbms