Column-oriented Database Systems : A Comparison Study

Arjun G. Roy, Mohammad K. Hossain, Arijit Chatterjee, William Perrizo
Department of Computer Science
North Dakota State University
Fargo, ND, 58102, USA
{arjun.roy, mohammad.hossain, arijit.chatterjee, william.perrizo}@ndsu.edu

Abstract

Over the years, there has been a slow but increasing focus on the column-oriented database management systems (DBMSs). Column-oriented DBMSs are different from the traditional ones in the way data is stored and accessed. Traditional ones store data row-wise while the column-oriented ones store the data column-wise. This allows column-oriented DBMSs to have extensive usage in various data warehouse applications because of their better performance in terms of read I/O in comparison to the conventional DBMSs. This is primarily due to the fact that column-oriented DBMSs only retrieve the columns defined in the query rather than the entire row in case of traditional DBMSs. In this paper, we study some of the existing and latest open-source column-oriented database management systems. Our approach is to highlight the salient features unique to the system. We also compare the DBMSs on the basis of some integral database features such as indexing, compression, transaction handling, etc. In addition, we give a concise but comprehensive summary of the commerical DBMSs using column-oriented architecture.

1 INTRODUCTION

Column-oriented database systems have come a long way since the initial work done on Decomposition Storage Model [10]. In recent times, column-oriented, also called as vertical databases have gained wide popularity in various applications including data warehouse, real-time analytics solutions, large-scale scientific data management, business intelligence, etc. This is attributed to their performance leverage over row-based DBMSs with respect to fewer read I/Os involved [1]. This underlying difference between the traditional row-based DBMSs and column-oriented vertical DBMSs can be explained using the Figure 1.

We acknowledge partial financial support for this research from a Department of Energy Award (award # DE-FG52-08NA28921).

Consider a *Student* table containing attributes *name*,

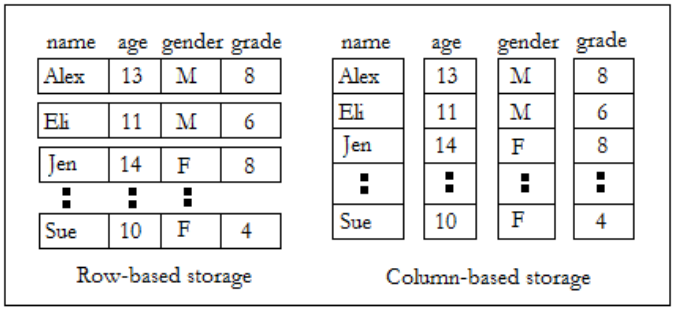

Figure 1: Row-based Vs Column-based

age, *gender* and *grade*. In a traditional database, the *Student* table is stored row-wise, one student record followed by another. In a vertical database, the table is vertically sliced, i.e., each attribute is stored in a separate individual file. Of course, each attribute file can be stored across several multiple files depending on the table/attribute size. Now consider the query, *find the names of the student whose age is more than 12*. This query requires two attributes, name and age from the *Student* table to return the result. Clearly, column-oriented database will perform better over row-oriented since only two attributes need to be brought to memory which incase of the latter would be the entire table. Additional significant performance gain is because of compression since columns of similar datatype can be compressed much better in comparison of rows which are a collection of attributes of different datatypes and sizes. As a result, in real-life scientific applications, where the number of columns are in the order of hundreds, vertical databases can have significant performance speedup over row-based database systems. In addition, they have several other advantages over conventional row-based databases including better consistency is absence of record-level locking, cheaper insertion and deletion operation, etc. [24].

In this paper, our aim is to survey and compare open-source column-oriented databases. The paper is

organized as follows: In section 2, we give a concise but comprehensive summary of the commerical DBMSs using column-oriented architecture. In section 3, we give a brief overview of the four open-source column-oriented databases, namely, C-Store, Infobright, InfiniDB and MonetDB. In the next section, we compare these databases on the basis of some integral database features such as indexing, compression, transaction handling, etc. We end the paper with a section on conclusion and future works.

2 COMMERCIAL APPLICATIONS

Column-oriented approach has been commerically utilized in many database warehouse applications. Majority of them work as a database and analytics solution running in a massively parallel environment. Aster Data *n*Cluster by Aster Data Systems is a massively parallel processing hybrid row and column database. It has an integrated analytics engine to provide extremely fast storage and analysis of relational and non-relational data [2]. Sungard's FAME or Forecasting Analysis and Modeling Environment is a database management and analytics solution which provides real-time and high-volume time-series data management along with various other related tools [23]. Kickfire, originally owned by Kickfire Inc. and later acquired by Teradata was a data analytics solution based on column-wise storage [14]. Infobright Enterprise Edition by Infobright is the commercial version of the Community Edition (discusssed in the next section). Though it integrates with the popular MySQL database, it uses its own company specific data storage and query optimization mechanism based on column-wise approach [13]. 1010data is a data analytics providing service company offering database of the same name. It provides a high performance column-store database service based on cloud computing [4]. ParAccel Analytic Database by ParAccel Inc. is again a massively parallel processing platform utlizing column-based database [17]. Sensage Event Data Warehouse uses clustered and columnar database to address heavy data traffic over long time frames [20]. Vertica Analytic Database is a grid-based system utilizing column-oriented storage organization [26]. SAP HANA uses both, a row-store and column-store relational engine [3]. P-Tree, a patented technology based on vertical striping of data is being used commerically by Treeminer Inc. [25][18]. Apart from the above, several other commerical column-oriented databases are available in the market including Sybase IQ [12], SAND CDBMS [19], etc.

3 OPEN-SOURCE DATABASES

In this section, we disuss four major open-source and column-oriented Database Management Systems, namely, C-Store, Infobright, InfiniDB and MonetDB. LucidDB is a DBMS of the same family but because of lack of technical details, we choose not to consider it in our study [15]. Our focus in this section is to highlight the salient and unique features of the databases and give a brief overview of the system in general.

3.1 C-Store

C-Store was a collaborative research project at MIT which has now been developed and commerically called as Vertica. It brought lot of novel and interesting features to the column-oriented architecture such as efficient packing of objects, use of overlapping projections to store the data, query optimizer and executor based on columns, etc. [22].

The simplified architecture of C-Store consists of only three components - small Writeable Store (WS) for inserts and updates, a large Readable Store (RS) for optimized read and a tuple mover to move records from Writeable Store to Readable Store. The queries have to use both the components to carry out an operation. For example, an update operation involves insert at WS and a delete at RS. We must note that both RS and WS implement the same physical DBMS design. They are only different in the storage representation because of different demands of read and write operations.

The data model of C-store is based on the implementation of projections. The projections are anchored over a table and contain one or more columns from one or more tables [22]. These projections are horizontally partitioned into many segments, each one identified by a segment identifier. As we will see later, this technique is somewhat similar to the one used by Infobright and InfiniDB. At the physical level, it makes sense to keep segments of the same projection as close as possible. To do this, C-store defines a *storage allocator* which not only ensures proximity of segments, but also of the WS and RS segments sharing the same key range. Since all the columns are stored in overlapping projections, only a covering set of projections is required to return a query result. C-store makes use of two objects, storage keys and the join indices to reconstruct the table from the columns.

The compression in C-store is carried out by an encoding scheme. It describes four types of encoding, Type 1 - Self-order, few distinct value, Type 2 - Foreign-order, few distinct value, Type 3 - Self-order, many distinct values and Type 4 - Foreign-order, many distinct values [22]. Another compression technique

used in C-store is described here [11]. Here, they list three categories, *storage optimization* for compression of single value (null), *lightweight* for compression of sequence of values and *heavyweight* for compression on array of bytes using standard LZO algorithm [11].

3.2 Infobright

Infobright* is a database and warehouse system available in commerical as well as free community edition. The key idea that sets this system aside from other vertical systems is the use of *Knowledge Grid* - a small metadata layer in place of the regular indexes. *Knowledge Grid* consisting of *Knowledge Nodes* are much smaller in size in comparison to the regular indexes and thus allow much faster as well as in-memory processing. The main functionality of *Knowledge Nodes* is to describe chunks of compressed data also called the *Data Packs* [21].

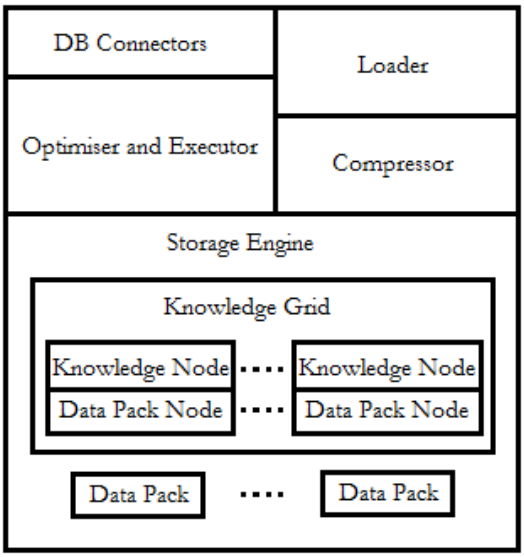

Figure 2: Infobright Block Diagram

Figure 2 shows a simple block diagram of Infobright. A detailed architectural diagram is given in [21]. In the diagram, the Connector module offers the interface to the database (ODBC, JDBC, PHP, etc.). The query optimization and execution are combined into a single module. This is logical considering that the query optimizer is dependant on the function of query executor over the *Knowledge Grid*. The *Knowledge Grid* consists of two metadata layers - *Data Pack Nodes* which store the statistical information about the Data Pack such as the max/min values, number of nulls, etc. and the *Knowledge Nodes* storing advanced information such as dependencies among the Data Packs, etc. A Data Pack is a row-wise grouping of 64K items of a column. The query optimizer and executor module as part of its functionality divides these Data Packs into three categories - Irrelevant, Relevant and Suspect [21]. Having this kind of classification gives lot of flexibility as well as efficiency. The Data Pack Nodes marked as Irrelevant can simply be ignored while querying. For example, say a query requests for data for atttibute 'a' above 50. Then the entire collection of Data Packs with max value below 50 will be ignored. [21] gives a detailed analysis of query processing. Overall, Infobright is an efficient column-store data warehouse solution with novelty in terms of *Knowledge Grid*.

* Infobright was previously called Brighthouse. We use the term interchangeably in this paper.

3.3 InfiniDB

InfiniDB is an efficient, multi-threaded analytic database system based on column-oriented storage architecture. Similar to Infobright, InfiniDB is available in two versions - Community Edition available free under GPL Licence and the commerical Enterprise Edition (scaled up version). InfiniDB is equipped with a comprehensive list of features [9]. In this paper, we only focus on some of the salient ones pertinent to the scope of this study.

InfiniDB uses an automated mix of vertical and horizontal partitioning. While the vertical partitioning allows faster processing by bringing only selected columns in the memory, a logical horizontal partitioning helps in reducing overall I/Os in horizontal and vertical direction. Because of this mix of partitioning, indexing is not required in InfiniDB. It further reduces I/O by making use of a small structure called Extent Map (explained later). InfiniDB allows massive parallel processing (MPP) thus achieving better performance with increase in the hardware. In an MPP environment where many nodes are involved, InfiniDB creates a large data cache to be accessed by the individual nodes in parallel for an otherwise non-sharing data architecture [9].

InfiniDB architecture consists of three components - *User Module*, *Performance Module* and *Storage Module*. Acting as an interface to the users, the top layer or the User Module is involved with taking the query requests from the users, distributing it to Performance Module and returning query result to the users. The Performance Module processes the query request by interacting with the Storage Module. The Storage Module is solely responsible for data storage. It may be noted that each of these modules can be one or many depending on the application and performance desired.

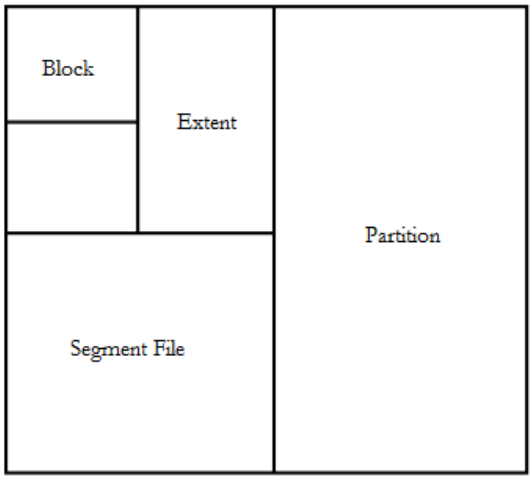

Figure 3: InfiniDB Storage Granularity

The storage mechanism in InfiniDB is quite different from other peer systems. It defines four levels of granularity - Block, Extent, Segment File and Partition. Block is the smallest unit equivalent of a page size (8K) represented by a Logical Block Identifier (LBID). Extent holds blocks to a size governed by the default number of rows for the column and the column datatype. Segment File represents the physical file on the disk. Lastly, Partition is a collection of Segment File. InfiniDB defines parameters that can be configured to control the count of these objects.

The concept of Extent Map though proprietary to InfiniDB is somewhat similar to Data Nodes of Infobright. It is basically a small structure stored inside the Segment File which catalogs all the Extents and Blocks through their LBIDs. It stores information such as the minimum and maximum value of Extent. Thus depending on the query requested, collection of Extents can be ignored and only the relevant Extents be used for processing. This allows significant I/O reduction and eliminates the use of indexing [9].

3.4 MonetDB

MonetDB is one of the most mature column-oriented database system. Developed by Centrum Wiskunde & Informatica, Netherlands, MonetDB has purely been a research project since its inception in 1995 [7]. Over the years, it has developed significantly in different aspects of database management system and currently hosts a family of products such as XQuery [6], MonetDB-GIS [8], etc. There are several features unique to MonetDB. A comprehensive study on its features is beyond the scope of this study. In this paper, we only focus on the MonetDB architecture including the storage model.

Each column in a MonetDB is represented by a two-column <surrogate, value> table also called the binary association table (BAT). The surrogate column is also called the head and the other column is called the tail. Thus BAT storage simply involves two columns, one each for the head and tail column. Figure 4 shows the BAT processing of MonetDB. The dotted columns represent the surrogates and the bold columns are the data stored using memory-mapped files. Use of memory-mapped files allows better I/O performance when the data is too large. The function of the BAT processor is to perform algebraic operations on BAT operators. For more information on BAT Algebra, please refer to [5]. The advantages of BAT operators stem from the fact that in case of traditional RDBMS, when involving Join / Select operator, there is a considerable real-time interpretation overhead [16]. The BAT Processor on the other hand breaks a complex expression into sequence of BAT Algebraic operators that perform a bulk processing of the entire column. In [16], authors show how BAT Algebra is benefitial in terms of compiler optimizations such as loop pipelining and CPU out-of-order speculation.

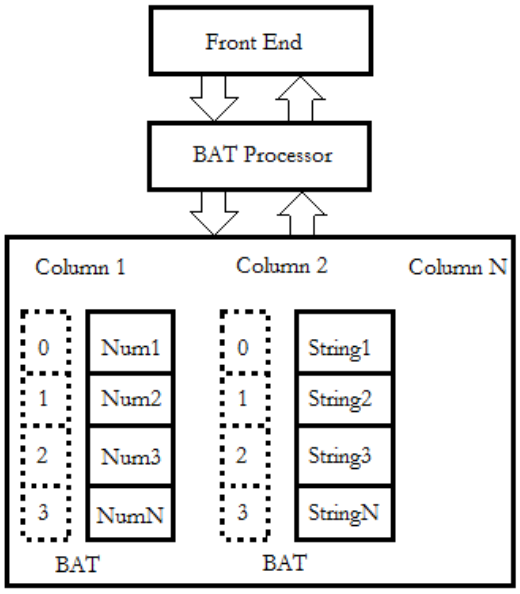

Figure 4: MonetDB: BAT Processing

The front-end of the MonetDB has been made flexible to accept a variety of languages such as SQL, XQuery, SPARQL, etc. Each of these languages is dealt differently by the BATs. For example, in case SPARQL which is a semantic web query language, MonetDB is implemented as a Resource Description Framework (RDF) database [16]. Overall, we conclude

that MonetDB exhibits features that are significantly different from those of the peer DBMSs.

4 COMPARISON MATRIX

In this section, we construct a Feature Vs Database matrix. Some of the features we cover are the indexing, transaction handling, kind of compression, interface to the database, etc. Certain features such as the Super-scalar compression in MonetDB, Snapshot Isolation in case of C-Store have not been discussed in this paper because of lack of space. We direct readers to the references for more details. Each database discussed has certain unique features but all of them are based on the concept of horizontal processing of vertical data rather than conventional vertical processing of horizontal data. The approach we found extremely interesting is the use of projections with redundant attribute columns to reconstruct the entire table in case of C-Store. The concept of Knowledge Grid consisting of the Knowledge Node and Data Pack is an innovative approach to avoid indexing in Brighthouse. Similar is the case with Extent Map in InfiniDB and the concept of version buffers for transaction handling. Lastly, MonetDB's role in the overall development of database system along with associated computer architectural innovations makes it one of the most mature and comprehensive column-oriented database management system.

5 CONCLUSION & FUTURE WORK

In this paper, we study some of the existing and latest open-source database management systems. Our aim is to highlight the important and unique features exhibited by the systems. We also provide a concise but comprehensive list of commerical database systems using column-oriented architecture. In future works, we plan to conduct a rigorous metric-based performance analysis on the systems by making use of standard benchmark tools and using data of varying size.

Feature Vs Database				
	C-Store	Infobright	InfiniDB	MonetDB
Indexing	Use of Column Projection. Join indexes for reconstruction.	No indexing. Use of Knowledge Grid.	No indexing. Use of Extent Map.	Just-in-time indexing.
Data Partition	Vertical Partition called Segments.	Vertical Partition using Data Packs	Vertical and Horizontal Partitioning.	Information unavailable.
Compression	Use of Encoding Scheme.	64K items vertically compressed in Data Packs.	Compression 65%-95% Real-time Decompression.	Super-scalar Compression. [27]
Unique DB Object	Projections.	Knowledge Grid, Data Packs	Extent Map.	Binary Association Table.
Transactions	Snapshot Isolation. [22]	Information unavailable.	Use of Version Buffer.	Based on Optimistic Concurrency Control.
Interface	Only read-only SQL queries.	MySQL	MySQL	MySQL, SPARQL, XQuery.
Commerical Use	Vertica	Infobright Enterprise Edition.	InfiniDB Enterprise Edition.	Pure Research Project.

References

[1] D. Abadi, S. Madden and N. Hachem. Column-Stores vs. Row-Stores: How Different Are They Really? ACM SIGMOD, pages 967-980, 2008.

[2] Aster Data nCluster: The First MPP Database with Applications Inside. www.asterdata.com/resources/assets/wp_Aster_Data_Applications_Within.pdf

[3] M. Bernard. SAP High-Performance Analytic Appliance 1.0 - A First Look At the System Architecture. www.sap.com

[4] R. Bloor. Big Data Analytics - This time it's personal. 1010data.com/downloads/big-data-analytics.pdf

[5] P. Boncz. Monet: a next-generation database kernel for query-intensive applications. Ph.D. thesis. Universiteit van Amsterdam, 2002.

[6] P. Boncz, T. Grust, M. Keulen, S. Manegold, J. Rittinger and J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. ACM SIGMOD, pages 479-490, 2006.

[7] P. Boncz and M. Kersten. Monet: An Impressionist Sketch of an Advanced Database System. Proceedings Basque International Workshop on Information Technology, San Sebastian, Spain, July 1995.

[8] P. Boncz, W. Quak and M. Kersten. Monet and its Geographical Extensions: A Novel Approach to High-Performance GIS Processing. In Proceed-

ings of the International Conference on Extending Database Technology, Vol. 1057, 1996.

[9] Calpont InfiniDB Concepts Guide. www.calpont.com/phocadownload/ documentation/2.0.0/CalpontInfiniDBConceptsGuide_20-1.pdf

[10] G. Copeland and S. Khoshafian. A decomposition storage model. ACM SIGMOD, pages 268-279, 1985.

[11] M. Ferreira. Compression and Query Execution within Column Oriented Databases. M.S. thesis. MIT, 2005.

[12] P. Howard. Enabling in-database advanced analytics with Sybase IQ. www.sybase.com

[13] Infobright. www.infobright.org

[14] Kickfire. www.teradata.com

[15] LucidDB. luciddb.org

[16] S. Manegold, M. Kersten and P. Boncz. Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct. VLDB, pages 1648-1653, 2009.

[17] The ParAccel Analytic Database: A Technical Overview. paraccel.com/the-paraccel-analytic-database-a-technical-overview/

[18] W. Perrizo, Predicate Count Tree Technology, Technical Report NDSU-CSOR-TR-01-1, 2001.

[19] SAND. www.sand.com

[20] Sensage. www.sensage.com

[21] D. Slezak, J. Wroblewski, V. Eastwood and P. Synak. Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries. VLDB Endowment, pages: 1337-1345, 2008.

[22] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran and Stan Zdonik. C-Store: A Column Oriented DBMS. VLDB, pages 553-564, 2005.

[23] SunGard FAME. www.sungard.com/fame

[24] P. Svensson, P. Boncz, M. Ivanova, M. Kersten, N. Nes and D. Rotem. Emerging database systems in support of scientific data. Scientific Data Management: Challenges, Technology, and Deployment, pages 235-277 [Book chapter]

[25] Treeminer. http://treeminer.com

[26] Vertica. www.vertica.com

[27] M. Zukowski, S. Heman, N. Nes and P. Boncz. Super-Scalar RAM-CPU Cache Compression. 22nd International Conference on Data Engineering, 2006.