ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Volume 2, Issue 7, July 2013

Impact of Column-oriented Databases on Data Mining Algorithms

Prof. R. G. Mehta[1], Dr. N.J. Mistry[2], Dr. M. Raghuvanshi[3]

Associate Professor, Computer Engineering Department, SV National Institute of Technology, Surat, India [1]

Professor, Civil Engineering Department, SV National Institute of Technology, Surat, India [2]

Principal, Rajiv Gandhi College of Engineering and Research, Nagpur, India [3]

Abstract: Traditional data storage is row oriented and ideal for write sensitive transaction process but they are not suitable for many read sensitive analytical processes. The Data Mining algorithms are analytical in nature and dig the hidden information from the well of structured/unstructured data. They are more analytic, deal with read/search/lookup process for data aggregation, will be potentially enabled by column oriented data storage rather than traditional row oriented storage. In column-oriented database systems (Column store), each database columns are stored separately in contiguous manner, compressed, and densely packed, as opposed to traditional database systems that store entire records (rows) one after the other. In this paper we review the architecture of various open sources column oriented databases like InfiniDB, Monetdb and Infobright. We have compared performance of column store over row stores for the simple tree based classification algorithm and CAIM discretization algorithm. The Novel rule based storage structure for the classification model is proposed, posses simple and efficient way of storage and access. Superior performance of the algorithm with column-stores, have answered the CPU utilization issues for such large-scale data-intensive applications.

Key Words: Data Mining (DM), OLAP, OLTP, Column store, Row store, Classification, ID3, Discretization, CAIM

I. INTRODUCTION

The Major Database applications are divided in two worlds: OLTP and OLAP. Database normal applications are OLTP but the huge volume and vast information of the data has created a need of data warehouse for storage which stores analytical information rather than transactions [1]. The Decision Support systems (DSS) and many AI applications need extraction of important and useful hidden pattern in the form of information from this huge data warehouses and very huge databases. Many of these applications use DM algorithms, which are analytical in nature and they dig the hidden information from the well of structured/unstructured data [2]. They are more analytic and deal with read/search/lookup processes for data aggregation, will be potentially enabled by column oriented data storage rather than traditional row oriented storage [3]. In column-oriented database systems (Column store), each database columns are stored separately in contiguous manner, compressed, and densely packed, as opposed to traditional database systems that store entire records (rows) one after the other [4]. The key shortfall of column store is that they are not designed for data that changes often and individual record appending is not a strong suit. Rather they are designed to quickly compress, analyze and load large amounts of data that will remain static [5].

DM algorithms are categorized mainly in three groups: Classification/prediction, Clustering and Association mining algorithms. The ID3 algorithm is a famous decision tree based classification algorithm which classify the objects in predetermined categories by testing the values of their properties. It builds the tree in a top down fashion, starting from a set of objects and a specification of properties. At each node of the tree, a property is tested and the results used to partition the object set. The recursive process is repeated till the set in a given sub tree is homogeneous with respect to the classification criteria. In other words it contains objects belonging to the same category which will be a leaf node of the tree. At each node, the property to test is chosen based on information theoretic criteria that seek to maximize information gain and minimize entropy [2]. In simpler terms, that property is tested which divides the candidate set in the most homogeneous subsets.

In this paper we have compared performance of column store over row stores for the simple tree based classification algorithm. The Novel rule based storage structure for the classification model is proposed, posses simple and efficient way of storage and access. The performance of the famous CAIM discretization algorithm is tested with row and column oriented databases. The superior execution time of the algorithms with column

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Volume 2, Issue 7, July 2013

stores has answered the CPU utilization issues for such large-scale data-intensive applications.

II. BACKGROUND AND PRIOR WORK

In this section, we briefly present major important features of different column-store and performance relative to traditional row-stores. The idea of vertically partitioning and managing database tables to improve performance has been proposed in many literatures [6] [7] [8].

Monetdb is an open-source column oriented DBMS, developed at the database group of CWI over the past two decades [9]. Monetdb system pioneered the design of modern column-oriented database systems and vectorized query execution.

In Monetdb every n-ary relational table is represented as a collection of Binary Association Tables called BATs without any hole [9]. For a relation R of k attributes, there exists k BATs, each BAT storing the respective attribute as (key, attribute) pairs. The 'key' is system generated and identifies all attributes of the relational tuple. When a query is fired, the relevant columns are loaded from disk to memory but are glued together in a tuple N-ary format only prior to producing the final result. Intermediate results are also materialized as temporary BATs in a column format, which can be efficiently reused by recycling process of the optimizer. SQL queries are translated by the compiler and the optimizer converts them into a query execution plan that consists of a sequence of relational algebra operators. One or more Monetdb Assembly Language (MAL) instructions will be generated for each relational operator. Each MAL instruction performs a single action using one or more columns in a bulk processing mode.

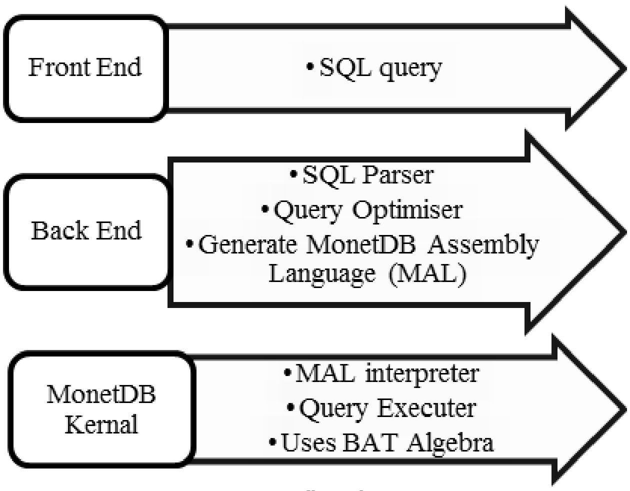

Fig.1 Monetdb Architecture

Infobright is designed to capture essential characteristics of the data: maximum values, minimum values, averages, deltas; whichever attributes can be logically inferred from the data itself [10]. This will make Infobright faster to execute the queries. The basic framework of Infobright is shown in Fig.2

The top layer is Knowledge grid which contains sets of knowledge nodes which are created at the time of initial loading of data. The data grids consist of 64k data packs aligned in traditional two-dimensional tables and the storage grids are completely hidden from the end user where the data persists. Depending upon the query and the information in first two layers, the analytical engine will first try to generate an acceptable answer by querying only the knowledge nodes, after which more specific queries can be posed, that will access the detailed data in the data grids. Infobright will use character map to store common strings with the vertical axis, and the frequency and location of those strings will be stored down the horizontal axis. When search on particular string is performed, common string will be searched in the knowledge nodes contain a significant number of those instances. Thus, Infobright can rule out large portions of the database that do not apply to the particular query. The distributed load processor of Infobright enables the product's rapid-fire loading approach. As the knowledge nodes are stored separately in the database, appended to their corresponding data packs, they can be queried without disturbing that data pack themselves. As inference and compression processes are separate from the load process, the lock times are minimized, which increases query speed and overall performance.

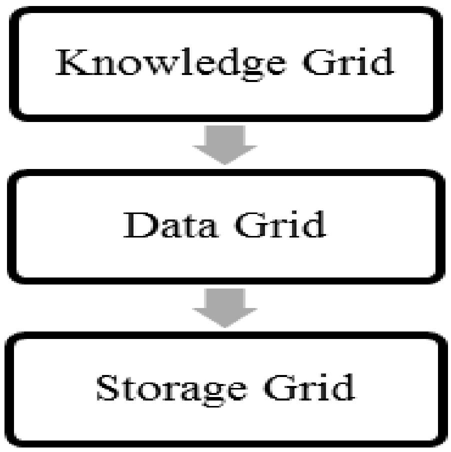

Fig 2 Framework of Infobright

InfiniDB is a multi threaded column oriented architected, make it more suitable for modern hardware that is multi-CPU/core based [11]. It also uses a form of logical horizontal range partitioning that does not require special storage placement or schema design. Thus as InfiniDB uses both vertical and logical-horizontal range partitioning, the I/O is reduced in both directions (column and row) and need of indexing is vanished. Other than the architecture advantages , InfiniDB posses many other database administrative advantages like superior

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Volume 2, Issue 7, July 2013

concurrency, ACID-compliant transactional support along with deadlock detection, crash recovery, multi-version concurrency control, platform portability etc. InfiniDB utilizes MySQL for its basic user interface makes it easy to use for MySQL users [11].

The performance of column oriented databases is tested using famous ID3 algorithm. CAIM is a very efficient discretization algorithm used as one of the precious preprocessing task for classification. It has also proven to be efficient online discretization for the tree based classification [2]. Next section describes ID3 algorithm in brief with proposed rule based storage for the classification model. The CAIM algorithm is also discussed in brief in the following section.

III. Data Mining Algorithms

A. *ID3 : A Tree Based Classification Algorithm*

ID3 is a famous tree based classification algorithm [2]. We have implemented simple tree based classification algorithm which uses Information gain as feature selection criterion and model is stored in the rule based manner. The process searches through the attributes of the training instances and extracts the attribute that best separates the given examples. If the attribute perfectly classifies the training sets then the process stops; otherwise it recursively operates on the n (where n = number of possible values of an attribute) partitioned subsets to get their "best" attribute. The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices. Simple ID3 is implemented for the efficiency analysis of the different databases.

- The datasets are preprocessed to posses certain qualities:
- Attribute-value is expected to be categorical
- Every instances of training/test dataset must have predefined classes
- Class attribute must contain discrete values
- Since inductive generalization is used (i.e. not provable) there must be enough cases to prepare accurate classification model and to test the validity of the model.

IV. Proposed Rule Based Storage Structure For The Classification Model

The rules generated using simple ID3 algorithm is stored in the file. As large dataset will produce huge set of rules, special storage structure is developed for the rule-base to fasten the search process. The rules are stored with suitable key which will guide the process for the next rule to be followed. This direct access of the next applicable rule will improve the test phase and the application phase of the classification process. The rule is stored in the following structure:

R: Feature Id(F), Feature value(V), N, Conclusion (C)
Will be read as : If F = V than C else goto next N line

Where N = -1 : No next possible rule available
N >0 : Relative line number to be followed.

For example consider a database "ZOO" having 50 instances and logical attributes Egg, Tail, Aquatic, Feathers, Breathes and Class. Sample rules generated while running zoo using the simple ID3 algorithm are listed below:

- Eggs NO 1 Mammal
- Eggs YES -1 Tail NO 3 Breathes YES 2 Aquatic NO 1 Insect
- Aquatic YES -1 Frog
- Breathes NO -1 Shellfish
- Tail YES -1 Feathers NO 1 Fish
- Feathers YES -1 Bird

As one of the characteristic of the attribute is to be a categorical in nature, discretization is one of the important preprocessing task, which convert the quantitative data in to categorical data. We use CAIM discretization for our implementation.

V. The CAIM Discretization Criterion

The Class Attribute Interdependency Maximization (CAIM) criterion is a heuristic measure that is used to quantify the interdependence between classes and the discretized attribute. It measures the dependency between the class variable C and the discretization variable D for attribute F, for a frequency distribution (quanta matrix) as shown in Figure 3, is defined as:

$$CAIM(C,D \mid F) = \frac{\sum_{r=1}^{n} \frac{\max_r{}^2}{M_{+r}}}{n}$$

Where n is the number of intervals, r iterates through all intervals, i.e., r = 1, 2, . . . , n, max_r is the maximum value among all q_{ir} values (maximum value within the r^{th} column of the quanta matrix), i = 1, 2, . . . , S, M_{+r} is the total number of continuous values of attribute F that are within the interval $(d_{r-1}, d_r]$ [5].

Class	Interval					Class Total
	$[d_0,d_1]$	....	$[d_{r-1},d_r]$	....	$[d_{n-1},d_n]$	
C_1	q_{11}	..,.	q_{1r}	....	q_{1n}	M_{1+}
:	....	....	....	....	....	....
C_i	q_{i1}	....	q_{ir}	....	q_{in}	M_{i+}
:	....	....	....	....	....	....
C_r	q_{s1}	....	q_{sr}	....	q_{sn}	M_{s+}
Interval Total	M_{+1}	....	M_{+r}	....	M_{+s}	M

Fig.3 Quanta Matrix

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Volume 2, Issue 7, July 2013

- The CAIM algorithm consists of these two steps:
- Initialization of the candidate interval boundaries and the initial discretization scheme and
- Consecutive additions of a new boundary that results in the locally highest value of the CAIM criterion.

Algorithm works in greedy manner. The discretization schemes generate high class-attribute interdependency and small number of discretization intervals.

VI. PERFORMANCE ANALYSIS

A. *Tree Based Classification*

Execution of the simple tree based classification algorithm is tested with Monetdb and Oracle11g. The dataset used for the analysis are listed in Table 1 and Table 2 depicts the execution time of both databases. The dataset are sourced from famous UCI machine learning repository [12].

Table 1: Dataset specification for ID3 algorithm

Sr.No	Data set	Number of instances	Number of features
1	Zoo	50	17
2	Covtype	2,000	56
3	Covbig	10,000	56

Table 2: Execution time of the classification algorithm

Sr.No	Dataset	Monetdb (Seconds)	Oracle (Seconds)
1	Zoo	5	1
2	Covtype	134	290
3	Covbig	2387	3324

Table 2, depicts the outperforming results of the algorithm with Monetdb compared to Oracle11g. For the Dataset "Covtype" (2000 instances and 56 features) Monetdb performance is almost two times faster compared to that with Oracle11g. Another interesting observation of Table 2 is for the dataset "zoo" (17 instances and 50 features). As smaller sized dataset needs less memory swapping, Oracle11g gives outperforming results.

VII. CAIM DISCRETIZATION

Execution of the CAIM algorithm is tested with Monetdb, Infobright (Column oriented database) and MySQL (Row oriented database) for different datasets. Execution time for CAIM discretization of continuous attributes of the datasets, described in Table 3 [12], is depicted in Table 4. As quantitative features are required to be discretized for the classification task, we will emphasis them for our CAIM algorithm. The missing values are replaced with mean values.

Table 3 Dataset specification for CAIM discretization analysis

Sr. No	Data set	Total instances	Total No. of Attributes	No. of Quantitative attributes
1	Iris	150	5	4
2	Forest Cover	5,81,012	54	10 (2)
3	Dermatolo gy	366	34	1
4	Credit Rating	125	16	5

Table 4 Execution time of the CAIM algorithm

Sr. No	Data set	Total CAIM discretization time (ms)		
		MySQL	Infobright	MontDB
1	Iris	1327	1380	791
2	Forest Cover	801024	792449	558296
3	Dermatolog y	315	296	172
4	Credit Rating	2591	2481	1563

The dataset "Forest Cover" is containing above 5.8 million instances, consumes more than 13 minutes in MySQL which is far larger than that of Monetdb, that is around 9.3 minutes. Here, Infobright executes the process around 1 minute less than that of MySQL.

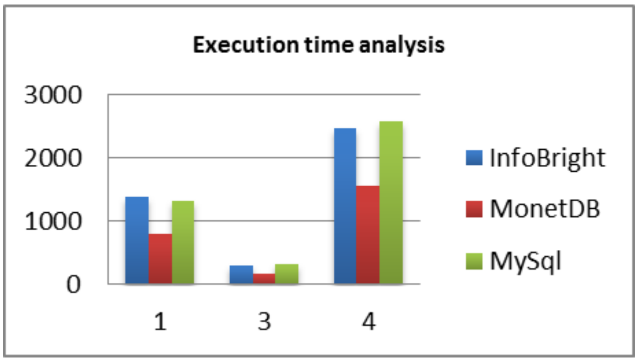

Fig 4: Execution Time comparison

From Figure 4, Monetdb is emerged to be faster among the remaining three databases. The grid based hierarchical storage structure of Infobright makes it slower than Monetdb. The graphical analysis of the execution time in Figure 4 depicts that for the large database with more number of columns, the execution with column store will be faster compared to row stores. In processing the small datasets, as less I/O task needed for data transfer and

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Volume 2, Issue 7, July 2013

data swapping, efficient CPU utilization will improve the execution time of the whole process.

VIII. CONCLUSION

In this paper, we attempted to describe the architecture of three famous datasets Monetdb, Infobright and InfiniDB. We compared the performance of Column oriented database (Monetdb) with Row oriented database (Oracle11g) using simple ID3: classification algorithms. The outperforming results of column oriented data for DM algorithms are encouraging. We also compared columned stores with MySQL, another famous row oriented database with the help of the very efficient discretization algorithm CAIM. The results of Monetdb is outperforming among other three databases, as the columned access for individual attribute will be faster in columned databases. The fast growing analytic applications and analytical DM algorithms initiatives would be a columnar database because of the faster response of the complex process, which emphasis more on columned access rather than whole row.

REFERENCES

1. H. Plattner, "A common database approach for OLTP and OLAP using an in-memory column database," in Proc. Of SIGMOD'09, pp. 1-2, 2009.
2. M. R. Lad, R. G. Mehta, D. P. Rana, "A Novel Tree Based Classification," IJESAT, Vol-2, pp. 581-586, 2012
3. 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, N. Tran, and S. B. Zdonik. "C-Store: Column-oriented DBMS," in Proc. of VLDB, pp 553–564, 2005.
4. Abadi, D.J., Madden, S.R., and Ferreira, M.," Integrating compression and execution in column-oriented database systems," in Proc. of the ACM SIGMOD, pp 671-682 , 2006.
5. M. Zukowski, P. A. Boncz, N. Nes, and S. Heman. "Monetdb/X100 - A DBMS In The CPU Cache," IEEE Data Eng. Bull., vol. – 28, issue 2, pp 17–22, 2005.
6. Abadi, D.J., Madden, S.R., and Hachem, N, "Column-stores vs. row-stores: how different are they really?," in Proc. of the ACM SIGMOD'08,pp 967-980, 2008.
7. Harizopoulos, S., Liang, V., Abadi, D.J., and Madden S. "Performance tradeoffs in read-optimized databases," *in the proc. of VLDB*, pp. 487-498,2006.
8. Daniel J. Abadi, Peter A. Boncz, Stavros Harizopoulos , "Column-oriented Database Systems," *VLDB*, tutorial, p.5, 2009
9. The MonetDB website , http://monetdb.cwi.nl
10. The Infobright website, " http://www.infobright.com"
11. The InfiniDB website "http://www.infinidb.org"
12. The UCI machine learning repository, "http://cml.ics.uci.edu"