Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 6, June 2015, pg.966 – 974

RESEARCH ARTICLE

Association Rule Mining for Structured Data in Big Data Using Parallel RDB-Miner Algorithm

Tererai Tinashe Maposa[1], Manoj Sethi[2]
[1]Department of Computer Engineering & Delhi Technological University, India
[2]Department of Computer Engineering & Delhi Technological University, India
[1] tereraimaposa@gmail.com; [2] manojsethi@dce.ac.in

Abstract— In recent years data volumes have astronomically skyrocketed. Almost all data is now regarded as big data and is now measured in the magnitudes of Yotabytes and Petabytes. This has been catalyzed by the rapid development of the Internet and the central role it has claimed in our daily lives. Data has become the most valuable asset for most organizations as it allows them to glean some business intelligence from it. Commercial organizations now gather as much data about their clients as possible. Governments have also invested heavily in data gathering and analytics. This has resulted in an enormous sea of data of all formats and structures. However there has been a misconception that all big data is unstructured or semi-structured and do not have a defined structure. This has resulted in a bias towards research areas. Lately most technologies and techniques that have been developed are intended for unstructured and semi-structured big data. This paper aims to parallelize an association rule mining algorithm for relational data called RDB-Miner Algorithm and implement it on structured big data in a HadoopDB multi-cluster environment.

Keywords— Big Data, RDB-Miner algorithm, HadoopDB, SQL-on-Hadoop, association rule mining

I. INTRODUCTION

The role that the Internet has taken in our daily lives has birthed the era of big data. This has been aided by technological advancements in hardware; hardware capacities (storage and processor power) have been ever increasing whilst its cost has been plummeting. This has afforded the scientific and commercial communities the luxury of collecting any data they deem necessary. Data from sensor networks, meteorological departments, genomics, medical data (such as X-ray images), posts on social networks (such as photos and comments on Facebook and Twitter), advertisements, stock exchanges, retail transactions, online video and audio conferencing, and so on is now being stored with relative ease[1, 2]. This obsessive collection of data has resulted in a massive sea of "meaningless" data. Consequently, it has become a cumbersome task to mine/extract and discover any business intelligence, patterns or associations from this large pool called BIG DATA [3]. Many technologies and techniques have been developed in order to aid in the data mining process. However, most projects have been aimed at unstructured and semi-structured data. A blind eye has been turned on the structured big data. This project aims to enhance the RDB-Miner algorithm which is an association rule mining algorithm for relational data model (structured data). The project intends to make it applicable and adoptable to structured big data. The project aims to parallelize the algorithm and implement it in a multi-cluster HadoopDB environment.

II. STRUCTURED DATA IN BIG DATA

Structured data is data that has a defined, predictable and repeating layout. It is data that can be easily put into rows and columns or that can take up the relational data model. As the world's view of data transitions from just data to big data, there is a false impression that all big data is unstructured or semi-structured. There is a notion that only unstructured data is increasing and only it requires new handling techniques. The heat map by the McKinsey Global Institute et al [1] below shows that all sectors still produce more text or numbers (which is in most cases structured) than any other types of data (audio, video and images which are unstructured data). The heat map is in terms of the number of units and not size of data.

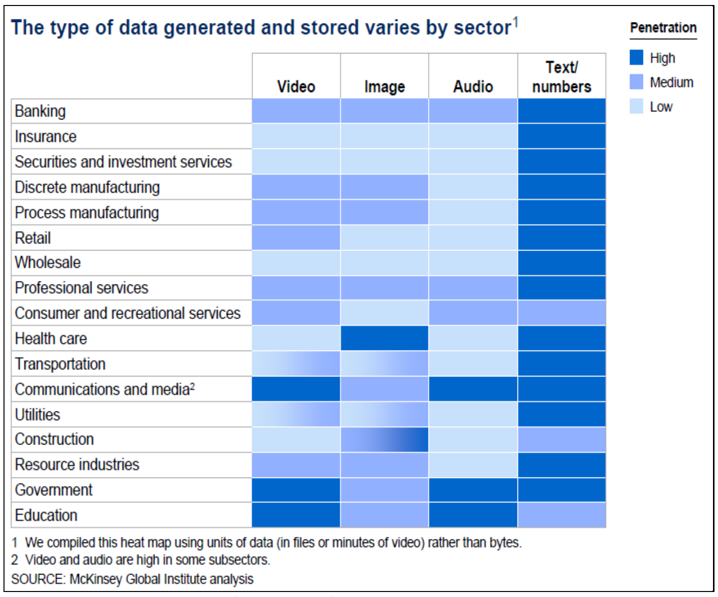

Figure 1: Heat Map for types of data generated by various sectors [1]

According to an article on IBM Data Magazine by Cristian Molaro et al [5], structured data still constitutes approximately 10% of the overall volume of big data. He also claims that this 10% is the one that matters the most because most of the 90% of the unstructured data are "cutie selfies" which in most cases have no business value. Most organizations, decision makers, and business analysts and other information consumers, can derive better business intelligence from big data–derived information when it is organized in a defined and structured manner rather than an unstructured one. This proves that as much as most of the big data (in terms of size) is unstructured, we cannot afford to ignore the structured data in big data. This 10% of big data requires new techniques and technologies because it is too big for traditional methods. There is therefore need for new platforms that can efficiently manage the enormous volumes of structured data being generated. Most algorithms that are currently in existence are mainly intended to handle unstructured data and may not be as efficient in handling structured data.

III. EXISTING TECHNOLOGIES FOR STRUCTURED DATA IN BIG DATA

Hadoop is synonymous to big data; it is difficult to talk about one and leave the other. However due to the steep learning curve involved in Hadoop and its inability to properly handle structured data; some hybrid technologies that combine the desired features of databases and Hadoop are being developed. These technologies are called SQL-on-Hadoop and they include, among others:

1) ***HadoopDB*** was developed at Yale in 2009. The thrust behind HadoopDB is to connect multiple single-node database systems using Hadoop as the task coordinator and network communication layer [6, 7]. Queries are parallelized across nodes using the MapReduce framework. HadoopDB runs on a shared-nothing architecture using commodity hardware. Any Open Database Connectivity (ODBC) compliant database can be used for example MySQL or PostgreSQL.
2) ***HAWQ*** (Hadoop With Query) is a commercial product developed by Pivotal HD which enables enterprises to benefit from tried and tested Massively Parallel Processing (MPP) based analytic features and its query performance while harnessing the power of the Apache Hadoop stack [8].
3) ***Spark SQL:*** Apache's Spark project is for real-time, in-memory, parallelized processing of Hadoop data. Spark SQL builds on top of it to allow SQL queries to be written against data.

Other tools that help support SQL-on-Hadoop include BigSQL, Apache Drill, Hadapt, H-SQL, Cloudera's Impala, JethroData, Polybase, Presto, Shark (Hive on Spark), Splice Machine, Stinger, and Tez (Hive on Tez).

IV. RDB-MINER ALGORITHM

Association Rule Mining (ARM) is one of the most prominent fields of data mining. It seeks to discover data items that frequently occur and their relationship. ARM has two steps involved: discovery of frequent itemsets and the establishment of association rules [9]. The classic algorithm for association rule mining, the Apriori algorithm, was first developed in 1994 [9]. The Apriori algorithm uses the layer search method to produce (k + l) itemsets from the k itemsets [9]. Many enhancements of the Apriori algorithm have been introduced; these include among others, Count Distribution [10], Data Distribution [10] and Candidate Distribution [10]. Most of them work on the market basket data format which is represented as a set of rows where each row consists of a unique identifier called transaction ID and a set of items purchased in the transaction. Market basket data violets the relational data model which does not allow multi-valued attributes.

Abdallah Alashqur in [11] introduced the RDB-Miner algorithm which directly mine association rules from a relational dataset. According to [11] RDB-MINER can be viewed as an algorithm that performs inter-attribute frequent itemset mining.

Algorithm *RDB-MINER [11]*

Input

```
R: a database relation
exclude_set: a subset of the attributes of R
0 Begin
1 Varchar SQL_str (512)
2 Compute_N (N, R, exclude_set)
3 Compute_PowerSet ( P (A) , R, exclude_set) ;
4 For c = 1 to N do
5 Extract_Ec (Ec , P (A) );
/* Ec ⊂ P (A) and each ISI ∈ Ec has a cardinality of c. */
6 For each itemset intension ISI ∈ Ec do
7 Generate_SQL (SQL_Str, ISI, Relation_Name);
8 Execute SQL_Str;
9 SQL_str = “”;
10 End
```

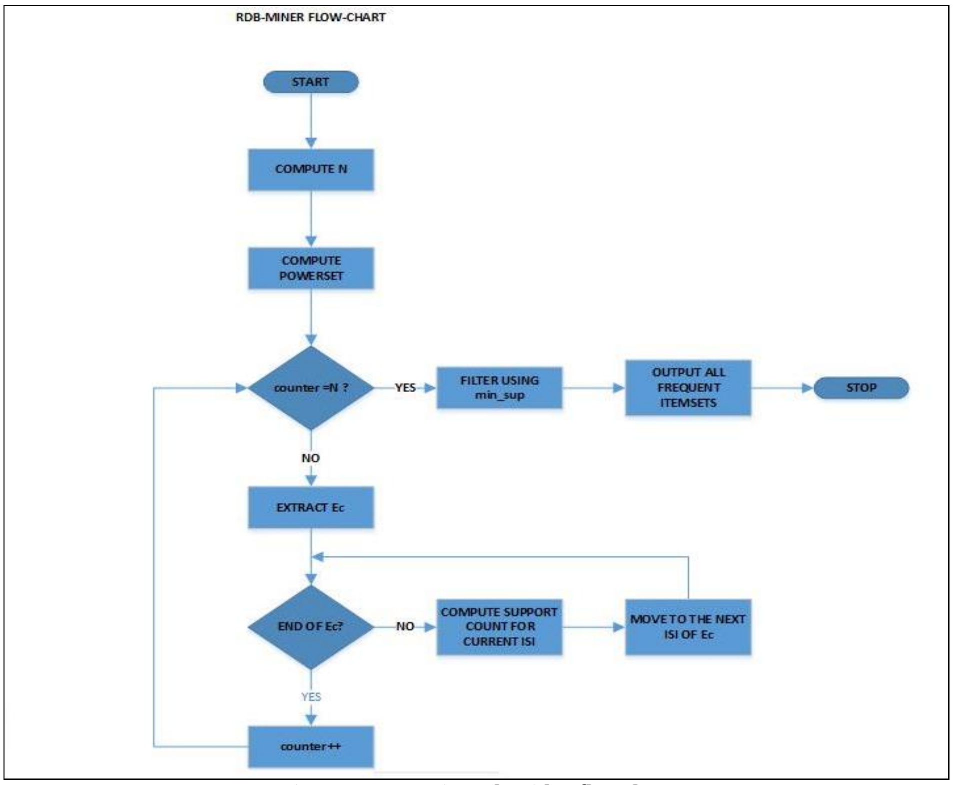

Figure 2: RDB-Miner algorithm flow chart

V. PARALLELIZATION STRATEGY

To parallelize the RDB-Miner algorithm we followed the logic outlined in [12]. However the proposed algorithm will not broadcast the consolidated global frequent itemsets to all nodes in the cluster; instead each node will transmit its local frequent itemsets to the coordinator which in turn consolidates the local findings of each node to construct the global frequent itemsets. The coordinator will filter for frequent itemsets using the supplied min_sup. This strategy seeks to reduce node communication hence making the algorithm more efficient as outlined in [13, 14, 15]. Nodes will only have to communicate with the coordinator node.

The transaction data is horizontally divided into N data subsets which are sent to N number of nodes. We modified the second **for** loop of the RDB-Miner algorithm; such that before it moves to ***k-itemsets*** it has to consolidate the ***(k-1)-itemsets*** from all the nodes. The logic is as follows:

1. Each node scans its data subset to generate the set of candidate itemset ***C p*** (candidate ***k-itemset***);
2. ***N*** nodes respectively accumulate the support count of the same itemset to produce the local support, and determine the set of frequent itemset ***Lp*** in the partition. At this point the nodes will not use the min_sup parameter to filter the frequent itemsets.
3. Each node then sends its computations to the coordinator for aggregation/consolidation.
4. The coordinator merges the output of the N nodes and filters the merged result using the set min_sup to generate the set of global frequent itemset ***Lp***.
5. Do the same for ***(k+1) –itemsets*** until c=N (number of attributes in the relation).

The algorithm enforces the Apriori property which states that any subset of frequent itemset must be frequent [16]. It achieves this by ignoring all the infrequent itemsets in the preceding iteration (N-1) when it is executing iteration N. This technique not only reduces the average transaction length but also reduces the data set size significantly. The number of items in the data set might be large, but only a few will satisfy the support threshold [15].

1) Pseudo Code for the proposed PRDB-Miner Algorithm

Input at each node
R: a partition of database relation
exclude_set: a subset of the attributes of R
min_sup: user specifies the min support to be used by the coordinator node

```
0  Begin
1  Varchar SQL_str (512)
2  Compute_N (N, R, exclude_set)
3  Compute_PowerSet ( P (A) , R, exclude_set) ;
4 c = 1
5          While c < (N+1) do
6          Extract_Ec (Ec , P (A) );
7          For each itemset intension ISI ∈   Ec do
8          Generate_SQL(SQL_Str,ISI,Relation_Name);
9                    Execute SQL_Str = Cp;
10                   Send_Results(Cp);
11                   End For

12 If CheckNodeCount() = TRUE then ConsolidateResults();
13             c ++
14             GOTO 5
15         End While
```

The improvements were put on lines 4 through 12. The explanations are as follows:

- Lines 1 to 3 remain the same but they are executed at the master/coordinator node,
- Line 4: a counter, c is initialized to 1,
- The commands in line 5 to line 10 are executed at the slave nodes,
- Line 5: the **while** loop checks to see if the counter c which counts the number of attributes in the relation/table has reached the total number of the attributes in the relation. If it true then it exits the loop and goes to line 16 and lets the master display the results,

- Lines 6 to 9 are the same as in the original algorithm,
- Line 10: ***SendResults()*** is a function that sends node results to the coordinator node. Its argument is the result generated by the ***Execute_SQL*** function in line 9.
- Line 12: ***CheckNodeCount*** function is executed by the coordinator only to check if all slave nodes have sent their results to it. It returns TRUE if all nodes have sent else it returns FALSE.
- Line 12: ***ConsolidateResults*** function is also used by the coordinator only to combine the results from all nodes and filters for frequent itemsets using minSup
- Line 13 increments the counter c
- Line 14 triggers the agent at each node to execute the mining function at each node. It passes the incremented counter c as an argument.

Below is the flow chart of the proposed Parallel RDB-Miner.

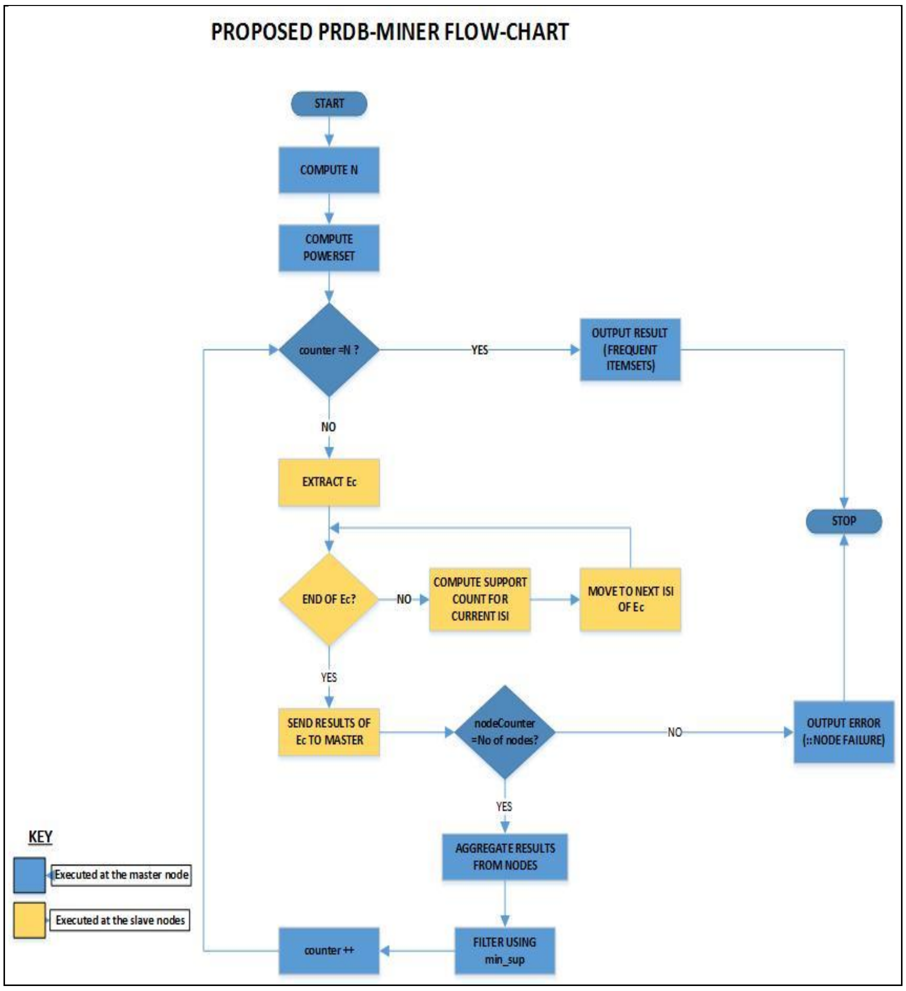
PROPOSED PRDB-MINER FLOW-CHART
START
COMPUTE N
COMPUTE POWERSET
counter =N ?
YES
OUTPUT RESULT (FREQUENT ITEMSETS)
NO
EXTRACT Ec
STOP
END OF Ec?
NO
COMPUTE SUPPORT COUNT FOR CURRENT ISI
MOVE TO NEXT ISI OF Ec
YES
SEND RESULTS OF Ec TO MASTER
nodeCounter =No of nodes?
NO
OUTPUT ERROR (::NODE FAILURE)
YES
AGGREGATE RESULTS FROM NODES
KEY
Executed at the master node
Executed at the slave nodes
FILTER USING min_sup
counter ++

Figure 3: Parallel RDB-Miner flow chart

VI. IMPLEMENTATION OF THE PARALLEL RDB-MINER ALGORITHM

We implemented the algorithm on a four node HadoopDB cluster. One of the nodes acted as the coordinator. The coordinator was responsible for aggregating node findings, filtering for frequent itemsets and displaying the final results. The coordinator was also the point of entry for the query where the user triggers the application. The coordinator also acted as the master node which hosted the NameNode and JobTracker; important components for the Hadoop framework. A simple recommender application was developed. The application used the PRDB-Miner algorithm to find product association based on customer reviews. The data used was formatted into three columns {userID, prodID, rating}. The application would find users who highly rated product X and also likes product Y. If the number of these users is high then product X and Y are highly associated to each other and that number of users is the support count for the rule X=>Y. The user interface for the application is web page designed using PHP. The user will be able to specify the minimum support from the proposed user interface. The relational database used on top of Hadoop is MySQL because it is ODBC compliant and also open source.

VII. EXPERIMENTAL RESULTS

In this paper we observed **Execution time** as a performance indicator. Execution time denotes the time taken (in seconds) by the application to produce all association rules with the desired support. The execution time was displayed and observed on the master node of the cluster. Three variables were altered as the performance was being observed. The variables that were altered are ***minSup***, ***number of cluster nodes*** and ***volume of data*** being worked on.

A. Results for varying minSup

This experiment was carried out with 1.7 million records horizontally partitioned and uploaded into three cluster nodes. The minSup variable used is the absolute count. The graphical results are shown below.

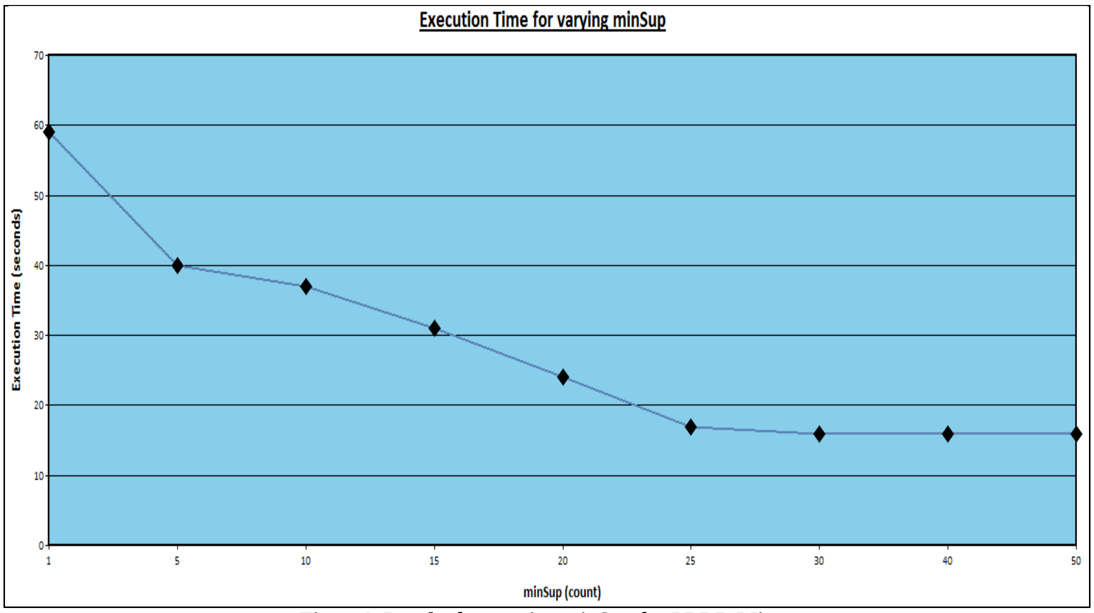

Figure4: Results for varying minSup for PRDB-Miner

Observation

The execution time gradually declined as the minSup increased. The execution time had a steady decent from minSup of 0 to 25 but it eventually became constant as the minSup continued to increase.

Explanation

Low minSup value means that the algorithm will have to deal with a lot of data hence the longer execution times. The master node is responsible for filtering for frequent itemsets using the minSup value; if the value is low it means a lot of itemsets will qualify as frequent itemsets. If the number of records remains high for most cycles of the algorithm, it results in longer execution time. However as the minSup increases, it means that the algorithm filters out most of the records and disqualifies them as infrequent itemsets. As the algorithm continues to scan the node databases it will be left with just a few records to deal with hence it becomes faster to generate the rules.
As the minSup continues to increase, the execution time becomes constant because the effects of all big minSup values will be the same. For example if there is no frequent itemset with a support of 30 in the whole cluster, if we put a minSup of 31 and if we put 50, the results will the same because no frequent itemset will be found in both scenarios.

B. Results for varying data volumes

The algorithm was also tested for varying volumes of data. This experiment was carried out using three nodes. Each node was allocated an equal potion of the whole data. The records in each partition were equally increased by 1 000 000 records during each experiment. The total number of records used in the five experiments was, 1.2, 1.5, 1.8, 2.1 and 2.4 million respectively. The minSup was set to default 5 during the whole experiment. Below is the graph for the execution time.

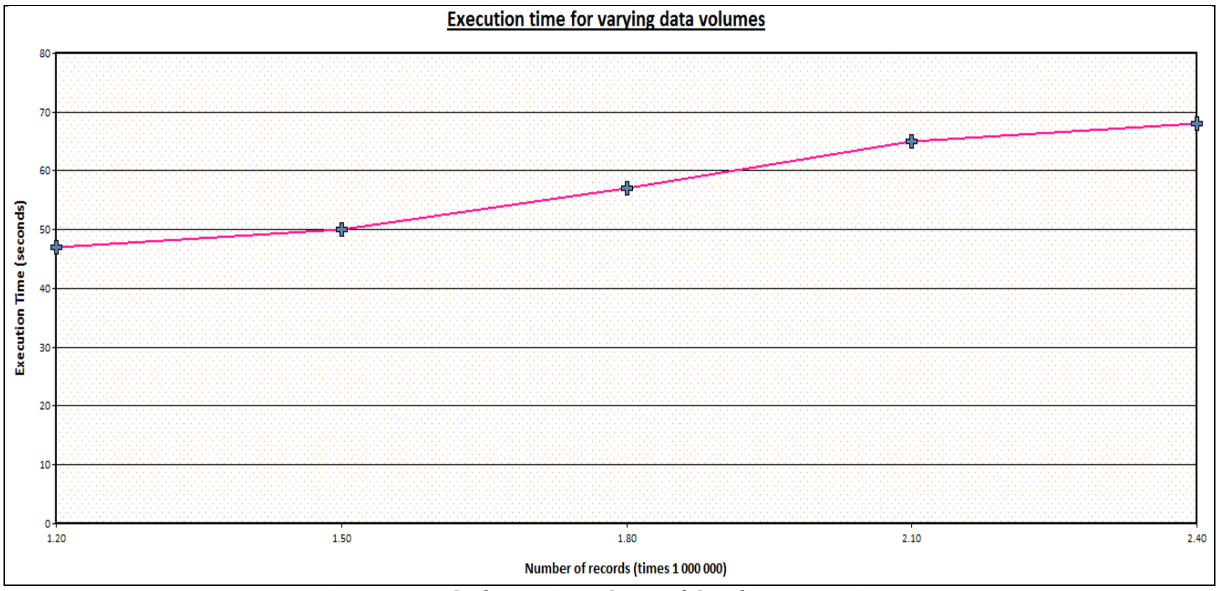

Figure 5: Results for varying volumes of data for PRDB-Miner

Observation

The execution time gradually increased as the volumes of data increased. However the increase was not very significant; an average of 2 to 3 seconds between experiments was observed.

Explanation

As the volume of data increases it means more data items and consequently more frequent itemsets. The more data there is the higher the execution time because the number of records remains high for most cycles of the algorithm. More records imply that large volumes of data will be exchanged between slave nodes and the master node resulting in some overheads. Also there will be strain on the master node as it will be having more data to filter and aggregate.

C. Results for varying number of cluster nodes

The algorithm was also tested on varying number of nodes in the cluster. The maximum number of nodes we could use was four. The number of records and minSup remained constant in this experiment. 1.7 million records and minSup of 5 was used in this experiment. Below is the graph of results observed:

Observation

The execution time gradually decreased as the number of nodes increased.

Explanation

The steady and gradual decrease of the execution time is as a result of increased “division of labor”. Since the number of records remained constant as the number of nodes increased; it means each node received a smaller portion of the data than in the preceding run. Consequently this resulted in a shorter processing time for each node.

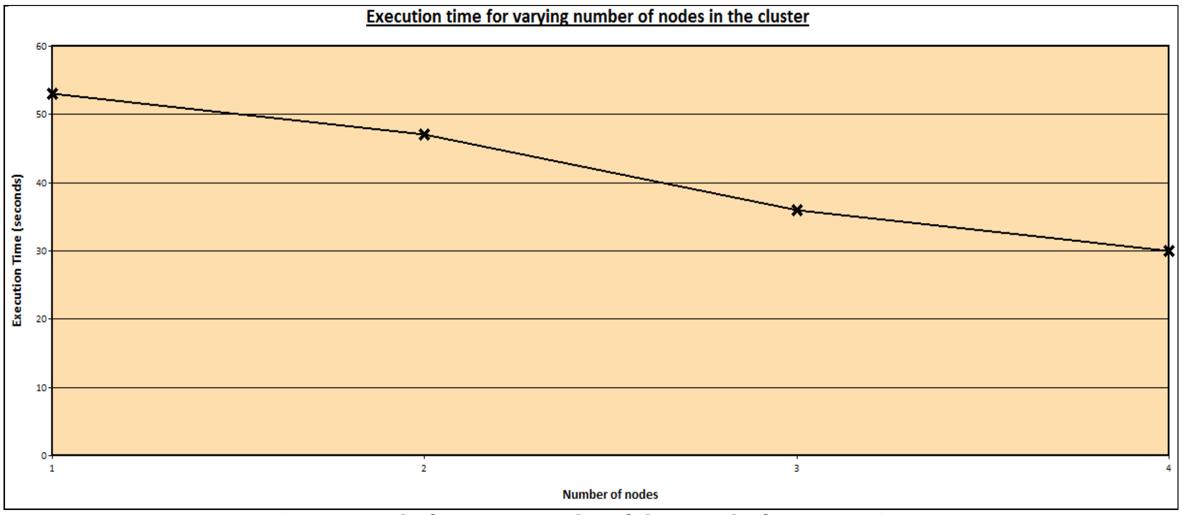

Figure 6: Results for varying number of cluster nodes for PRDB-Miner

VIII. OPTIMUM NUMBER OF NODES

It is our hope that there exist an optimum number of nodes for the algorithm. The optimum point is where we can get the best performance of the algorithm. As the number of nodes continues to increase after reaching this point, it is most likely that the performance of the algorithm will start to fall because the data will be continually partitioned into smaller and smaller chunks such that each node will have very few records. Each node will have very little to do. The major task will be that of filtration and aggregation which are undertaken by the master node. This will result in a bottleneck at the master node since it will be faced with a lot of records to deal with. In such cases parallelization will be of no effect as one node (the master node) will be left with much of the work. The optimum number of nodes is dependent on the number of records being acted upon. For fewer records the optimum number will be smaller.

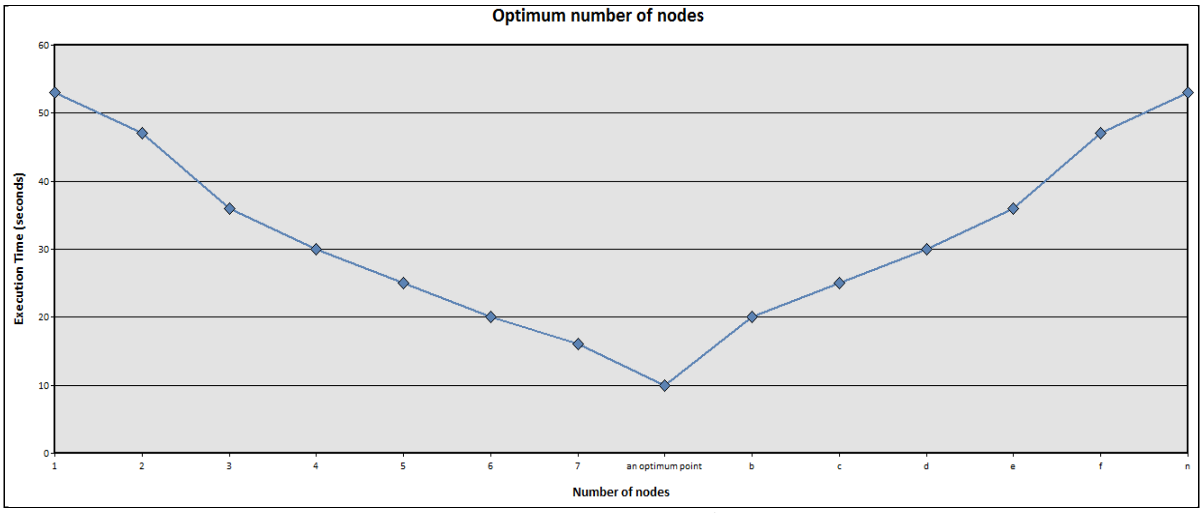

Figure 7: Optimum number of nodes

IX. CONCLUSION

The proposed algorithm, Parallel RDB-Miner performed remarkably. We also concluded that Parallel RDB-miner works well for relations that have a few number of attributes but have a large number of records. A high number of attributes results in a higher number of iterations thereby resulting in poor performance.
In future we hope to find a formula that calculates the optimum number of cluster nodes given the number of records. We also desire to further enhance the algorithm such that it becomes more resilient to node failure since this is a reality we cannot afford to ignore. Also we intend to test the algorithm on other open source SQL-on-Hadoop tools.

REFERENCES

[1] Marko Grobelnik, Jozef Stefan Institute, Ljubljana, Slovenia*, "Big Data Tutorial",* May 2012.

[2] Andrew Pavlo, Erik Paulson, Alexander Rasin, *"A comparison of approaches to large-scale data analysis,"* in Proc. SIGMOD'09, 2009, p. 165.

[3] Daniel J. Abadi, *"Data management in the cloud: limitations and opportunities,"* Bulletin of the IEEE Computer Society Technical Committee on Data Engineering.

[4] Xindong Wu, Fellow, IEEE, Xingquan Zhu, Senior Member, IEEE, Gong-Qing Wu, and Wei Ding, Senior Member, IEEE, *"Data Mining with Big Data",* 2014.

[5] *http://ibmdatamag.com/.*

[6] Azza Abouzeid1, Kamil BajdaPawlikowski, Daniel Abadi1, Avi Silberschatz1, Alexander Rasin2, *"HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads"***,** Yale University, Brown University.

[7] HadoopDB Project. Web page.*db.cs.yale.edu/hadoopdb/hadoopdb.html.*

[8] http://blog.*pivotal.io/big-data-pivotal/products/why-mpp-based-analytical-databases-are-still-key-for-enterprises.*

[9] R. Agrawal and R. Srikant, *"Fast Algorithms for Mining Association Rules,"* Proceedings of the International Conf. on Very Large Databases (VLDB'94), 1994, pages 487-499, Santiago, Chile.

[10] Jianwei Li, Northwestern University; Ying Liu, DTKE Center and Grad. Univ. of CAS; Wei-keng Liao, *"Parallel Data Mining Algorithms for Association Rules and Clustering"*, Northwestern University; Alok Choudhary, Northwestern University.

[11] Abdallah Alashqur *"RDB-MINER: A SQL-Based Algorithm for Mining True Relational Databases"* Faculty of Information Technology Applied Science University Amman, JORDAN.

[12] Lingjuan Li, Min Zhang, *"The Strategy of Mining Association Rule Based on Cloud Computing"*, College of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China.

[13] Dr (Mrs). Sujni Paul, *"Parallel and Distributed Data Mining",* Karunya University, Coimbatore, India.

[14] Dr (Mrs).Sujni Paul, Associate Professor, Department of Computer Applications, *"An optimized distributed association rule mining algorithm in parallel and distributed data mining with XML data for improved response time",* Karunya University, Coimbatore, Tamil Nadu, India.

[15] Mafruz Zaman Ashrafi, Monash University, David Taniar, Monash University, Kate Smith, Monash University, *"ODAM: An Optimized Distributed Association Rule Mining Algorithm",* IEEE Computer Society Vol. 5, No. 3; March 2004.

[16] Jaiwei Han and Micheline Kamber, *"Data Mining Concepts and Techniques, Third Edition"* page 298**.**