Exploring HADOOP as a Platform for Distributed Association Rule Mining

Shravanth Oruganti, Qin Ding, Nasseh Tabrizi
Department of Computer Science,
East Carolina University,
Greenville, North Carolina, USA
o.shravanth@alumni.ecu.edu, dingq@ecu.edu, tabrizim@ecu.edu

***Abstract* - Association rule mining is one of the important data mining techniques. Association rule mining is used to discover associations between different items in large datasets. The Apriori algorithm for association rule mining forms the basis for most other association rule mining algorithms. The original Apriori algorithm runs on a single node or computer. This limits the algorithm's capability to run on large datasets due to the limited computational resources available. There have been various studies for parallelizing the algorithm. In this paper, Apache Hadoop was chosen as the distributed framework to implement the Apriori algorithm and to evaluate the performance of the algorithm on Hadoop. The Apriori algorithm was modified to be run on Hadoop. Performance analysis shows that Hadoop is a promising platform for distributed association rule mining.**

Keywords* - *Cloud computing; association rule mining; data mining; Hadoop.

I. INTRODUCTION

Business Intelligence has become an integral part of many successful organizations. Analyzing data and making decisions based upon the analysis is very important for an organization's growth. Data mining techniques help analyze the substantial data available to assist in decision-making. Among the most frequently used data mining techniques, association rule mining is a very important one. For example, in market analysis, association rule mining helps identify what items are purchased together by customers and generate interesting rules (depending on the measure of interestingness selected) based on the transactional data. Many approaches [2,3,10,12,14] have been proposed to mine association rules but a majority of them depend on the Apriori algorithm as the basis [2]. Due to huge size of the datasets, running these algorithms and mining association rules on a single computer is not very efficient because it is limited by the processor capacity, RAM, storage, and various other factors. Hence, it is necessary to develop distributed algorithms to perform association rule mining. Performing large-scale computing and data mining is one of the issues in future computing.

It was observed that most algorithms for association rule mining ran out of memory even for small datasets and a small support count. There are some parallelized algorithms for association rule mining, but all of them handle the communication in the network, load balancing or other distributed tasks [1,4,5,6,9,13,15,17,18]. The goal of this work was to see if there would be a distributed framework where data intensive tasks could be performed without the overhead of the programmer managing the distributed part of the system. Apache Hadoop was shown to be a good framework for this task [16]. Very little research has been done on implementing association rule mining algorithms on Hadoop, although there are a few studies on implementing association rule mining using cloud computing [7,11,16]. For this study, Apache Hadoop was chosen as the distributed framework to implement the Apriori algorithm [2] and to evaluate the performance of the algorithm on Hadoop.

The rest of the paper is organized as follows. Section II introduces the background related to association rule mining and Hadoop. Section III details the proposed framework, algorithm, and implementation. Section IV presents evaluation followed by the conclusion and future work in Section IV.

II. BACKGROUND

This section introduces the concepts of association rule mining, the original Apriori algorithm for association rule mining, previous approaches to parallelize Apriori algorithm, the MapReduce framework, and Hadoop.

A. Association Rule Mining

Association rule mining was originally proposed for Market Basket Analysis [2]. By searching for frequent patterns in transactional data sets, interesting associations and correlations between item sets in transactional and relational databases may be discovered.

Market Basket Analysis is a modeling technique based upon the theory that if you purchase a particular group of items, you are more likely to purchase another group of items. For example, if a customer purchases some pizza dough, it is more likely he/she will also purchase some pizza sauce. Placing them at the same aisle in the store or even bundling them together could help increase profits. Just the knowledge that customers often purchase certain items in groups could open up new possibilities in businesses.

In the aforementioned example, the set of items a customer purchases is referred to as an itemset, and market basket analysis seeks to find relationships between itemsets. Typically the relationship will be in the form of a rule, such as: customer purchases pizza dough → customer purchases pizza sauce.

 ISBN: 978-1-61208-272-1

There are two measures, support and confidence, which can be used to measure the interestingness of the rules. Support indicates the frequency of the occurring pattern while confidence measures the strength of the association.

B. Apriori Algorithm

The Apriori algorithm [2] has been the basis for many other algorithms in association rule mining. Subsequently, there were many modifications and other algorithms to mine frequent patterns in data [3,10,12,14]. A brief summary of the Apriori algorithm is given below.

The Apriori algorithm is an iterative approach to generate frequent itemsets based upon a user provided minimum support and confidence. Candidate itemsets of size k are generated based upon frequent itemsets of size k-1. The basic algorithm works as follows (Fig. 1):

```
Ck: Candidate itemset of size k
Lk: frequent itemset of size k

L1= {frequent items of size 1};
for (k = 1; Lk! =∅; k++) do begin
    Ck+1= candidates generated from Lk;
    for each transaction t in database do
        Increment the count of all candidates in Ck+1 that are
            contained in t
    Lk+1= candidates in Ck+1 with min_support
end
return ∪k Lk;
```

Figure 1. Apriori algorithm

A major drawback of this algorithm is the high I/O costs. The database needs to be scanned during each iteration, which is expensive. With huge datasets, this may consume significant system resources in order to scan and hold the transactions in memory.

C. Parallel versions of Apriori

The authors of the Apriori algorithm also proposed three parallelized versions of the algorithm to run on multiple nodes [1], including Count Distribution, Data Distribution, and Candidate Distribution algorithms.

The Count Distribution algorithm uses a simple principle of allowing redundant computations in parallel on otherwise idle processors to avoid communication. The basic idea is that each processor will scan the local data asynchronously in parallel to do the local counting, but the candidate itemsets that each processor counts are identical except during the first pass. At the end of each pass, they must synchronize to calculate the global counts. Additional details may be found in [1].

The main disadvantage of the Count Distribution algorithm is that it does not exploit the aggregate memory of the cluster. The Data Distribution algorithm attempts to address this issue. Unlike the Count Distribution algorithm, the Data Distribution algorithm counts mutually exclusive candidate items. Hence, as the number of nodes increases, more candidate itemsets may be counted in one pass. The major downside of this algorithm is the additional overhead of transmitting the local data along with the candidate itemsets.

The Candidate Distribution algorithm does not require data or candidate itemset communication between processors. This addresses some of the problems associated with count and data distribution methods. In the aforementioned two approaches, dependencies exist between the processors; if the load balancing and synchronization is not performed properly, the algorithm slows down considerably. Candidate Distribution eliminates this by removing the node communication. The basic idea is to divide the candidate itemsets and data among the nodes in such a way that each processor or node can perform the counting of candidates independently of the other processors. This division is performed after a certain number of passes, which is determined heuristically during the iterations.

During the tests performed by the authors, they found that the Count Distribution algorithm is the best way to parallelize the Apriori algorithm.

D. MapReduce Model

MapReduce [8] is a distributed software framework introduced by Google in 2004 to support distributed computing on large datasets on clusters of computers. MapReduce was derived from the mapreduce feature of functional programming languages but was adapted to distributed computing. It consists of two steps: Map and Reduce.

The Map phase consists of a Mapper class that receives an input (key, value) pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. It essentially emits all the key/value pairs containing the same key to the reducer.

The Reducer class contains the reduce function that accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values and writes them to the specified output.

The basic MapReduce architecture is shown in Fig. 2.

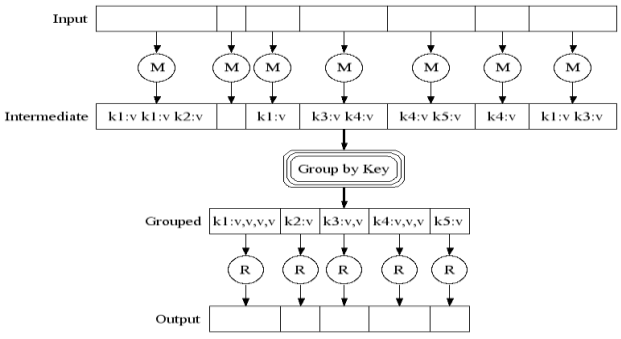

Figure 2. MapReduce architecture

To use the MapReduce programming model, each task must be represented as some computation on a list of key-value pairs.

E. Apache Hadoop

The following is a description of Hadoop from the official Apache website [20]: "Hadoop MapReduce is an open source software framework for writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner."

Hadoop MapReduce makes use of the Hadoop File System (HDFS) as the underlying distributed file system architecture. HDFS is responsible for distributing the data across multiple nodes and the number of replications, etc. There are many parameters which can be configured based on the requirement from the configuration files provided by the Hadoop distribution.

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. Further details may be found on the Apache Hadoop website [20].

Hadoop has three modes of operation: Standalone mode, Pseudo-Distributed mode, and Fully Distributed mode.

In Standalone mode, also known as local mode, there are no daemons running and everything runs within a single JVM (Java Virtual Machine). This is the default mode of operation and is suitable for running MapReduce programs during development, since it is easy to test and debug. In this mode, a single Hadoop node is created but does not use the Hadoop Distributed File System (HDFS). This means that all input and output files are read from and written to the underlying OS file system; thus, there appear to be no benefits in using HDFS.

In Pseudo-distributed mode, all of the Hadoop daemons run on the local machine, thus simulating a cluster on a single machine. Other than where the Hadoop processes are running (one machine vs. one machine per node), this mode is the same as distributed mode. In this mode, Hadoop starts all of the processes for all of the configured nodes on the same machine. This mode is useful because it allows the architect to observe how applications respond to running on a Hadoop cluster, but without the overhead of setting up the individual machines for the nodes of the cluster. While the task is much easier for a software architect using a cloud and Hadoop, there is still some overhead involved. Because the HDFS is used by default, benefits may be gained from using it.

In Fully distributed mode, the Hadoop daemons run on a cluster of machines. Each Hadoop node is started on the specified machine. As with pseudo-distributed, HDFS is used.

III. IMPLEMENTATION

The Count Distribution strategy was chosen for implementing the Apriori algorithm on a Hadoop cluster. Hadoop is not suited to implement the Data Distribution strategy since the data distribution cannot be controlled on Hadoop. It does not make sense in the MapReduce world to transmit data to other nodes. The Candidate Distribution strategy does not work either because a particular data block cannot be assigned to a particular node in Hadoop.

A very high-level view of the modified algorithm is provided below in Fig. 3.

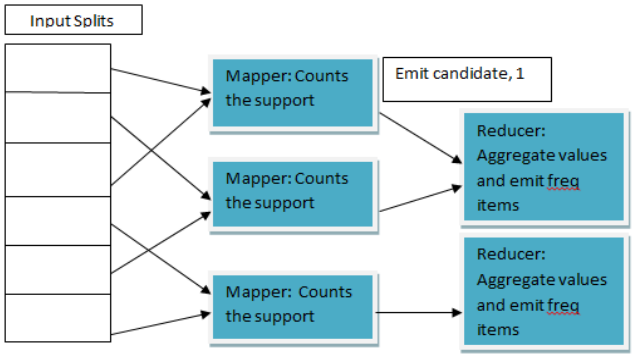

Figure 3. Design of the algorithm

The algorithm works as follows. By using the HDFS file system and configuring it, Hadoop can automatically split the data files into smaller chunks and distribute them over the Hadoop cluster. The replication factor can be specified to replicate the data on multiple nodes so that the algorithm works even if a node does not work during the execution. The output files containing the frequent itemsets are stored in the output folder of the HDFS. These files need to be retrieved to the local system to generate subsequent candidates.

For performing the counting in parallel, each map job is configured to have the candidate itemset list and each node in the cluster has certain data blocks. The jobs are run multiple times for each frequent itemset having the minimum support. Each Map task that runs on a node reads the chunks present locally and emits the candidate, a count of 1 for each time the candidate is found in the file. The reduce function then aggregates all these counts for a particular key and writes the candidate to the output file only if it satisfies the minimum count. The output file is then read and the candidate list for the next iteration is generated. Thus, the counting step is parallelized.

IV. EVALUATION

To evaluate how well the design of the algorithm scales on multiple nodes and whether Hadoop as a framework is a good fit to perform association rule mining, some analyses were performed. The following section details how the data were generated and various analyses performed.

 ISBN: 978-1-61208-272-1

A. Datasets used and Data Generation

We generated many datasets with varying parameters, but for the analysis we used four of them. We used a synthetic data generator to generate the transactions to perform frequent pattern mining. But the generated data would not be a perfect way to test the system as it was arbitrarily generated and may not really reflect real world data. Hence, we also used a dataset from the FIMI repository [19]. The dataset and a brief description of the dataset are given below.

1) Dataset: Accidents.dat

This dataset of traffic accidents was originally obtained from the National Institute of Statistics (NIS) for the region of Flanders (Belgium) for the period from 1991 to 2000. In total, 340,184 traffic accident records are included in the dataset with 572 different attribute values. On average, 45 attributes are filled out for each accident. More details about the attributes can be found in the FIMI repository.

2) Synthetically generated Datasets

The authors of the Apriori algorithm at IBM have developed their own tool to generate datasets to test their algorithm. They used different parameters to generate the dataset. The major factors are:

- Average size of the transaction (Ts)
- Number of transactions (Tn)
- Average size of maximal potentially frequent itemsets (P)
- Number of maximally potentially frequent itemsets (L)
- Number of items (N)

Many other studies used the aforementioned IBM Data generator to generate synthetic data generator. However the tool is now obsolete. Hence, ARtool[21], another open source tool package, was chosen for generating the synthetic data. The tool is an open source tool developed at the University of Massachusetts at Boston and it provides options to mimic the parameters and datasets. By using this tool, four datasets were generated. They are summarized in Table I:

TABLE I. DATASETS

Dataset	Tn	Ts	L	P	N
200K_f50	200000	50	2000	6	1000
200K_f30	200000	30	2000	6	1000
200K_f20	200000	20	2000	6	1000
100K_50	100000	50	2000	6	1000

B. Cluster Setup

Hadoop, as explained earlier, runs in three different modes. All three modes of operation were used while developing and testing the algorithm.

Standalone mode: When the Hadoop and MapReduce framework were tested, many test programs had to be written to test Mappers and Reducers performing tasks in parallel. For the purpose of testing the basics of Hadoop, the NetBeans IDE with a plug-in called Karmasphere was used to test Hadoop applications that had been written. Karmasphere simulates Hadoop and provides Mappers and Reducers within a single node. It uses only the local file system and no distribution of data occurs. The first basic version of Apriori was developed according to MapReduce framework in this mode and was tested on small datasets to determine if the core algorithm was generating correct results.

Pseudo-Distributed Mode: This mode uses the HDFS system and distributes data albeit in a single node. It simulates the Hadoop environment on a single node. Essentially it is a cluster with just one node in it. The Mapper and Reducer run as separate daemons in the system. This mode was used to test whether the program was reading and writing files correctly from and to the HDFS and local file system and vice versa.

Fully Distributed Mode: This is the mode where we have a full Hadoop cluster with multiple nodes running the program. Initially a Hadoop cluster was built with two nodes, i.e., a system and a VM within the system to test the application. We used this to test if the cluster was setup correctly and to understand the complexity of setting up a Hadoop cluster. The IBM smart cloud was then chosen to test the algorithm and the framework. IBM smart cloud is a PAAS (Platform as a Service) that provides Hadoop master and data nodes. The programmer does not need to create the cluster nor perform installation and maintenance of the cluster. This is automatically performed by the cloud provider. With access to 5 nodes in the cloud, multiple tests were performed to generate some interesting and promising results. The following sections describe those results.

C. Analysis on Number of Nodes vs. Running Time

In this analysis, the goal was to test if there would be an improvement in the running time as the number of nodes increased. The accident.dat from the FIMI repository was used as a dataset and the program was run on 2, 3, 4, and 5 nodes. The results are shown below in Table II and Fig. 4.

TABLE II. NUMBER OF NODES VS. RUNNING TIME

Dataset	Support	Nodes	Time(min)
Accidents.dat	0.8%	2	197
Accidents.dat	0.8%	3	152
Accidents.dat	0.8%	4	107
Accidents.dat	0.8%	5	92

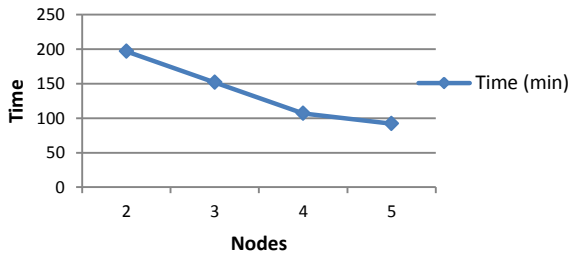

Figure 4. Number of nodes vs. running time

As is evidenced, as the number of nodes increases, the running time of the algorithm decreases drastically. But the decrease in time is not always at the same rate. This depends upon the size of the data, and more importantly, the split factor of the data. When the files are transferred from the local file system to HDFS, a block size needs to be specified for the file to be split into. Each block is then assigned to a node and all the operations on that block are performed by only that node. If the number of nodes exceeds the number of blocks, it is a waste of resources. So, at some point, increasing the number of nodes would not lead to the decrease in time. This point could not be reached in this analysis as there were a limited number of nodes in the cluster.

The important conclusion from this analysis is that running the Apriori algorithm parallel on individual data blocks on Hadoop does lead to a significant performance increase and Hadoop as a distributed framework is a good platform to use to perform data intensive computations. Another observation that should be noted is the advantage of using a cloud platform to achieve this. Both the individual cluster and the IBM smart cloud were utilized and clearly, in terms of adding new nodes to the cluster and scalability, using a cloud is much more advantageous than building one's own cluster. Substantial maintenance costs could be associated with maintaining one's own Hadoop cluster as typically such mining tasks are not run every day but rather once in two weeks or once a month. There are many cloud providers offering a Hadoop cluster and using these cloud vendors is much less expensive and also much more efficient than running it on one's own cluster.

D. Analysis on Running Time Comparison Based on Dataset

In this analysis, the goal was to test the performance of the algorithm on different datasets. The percentage decrease in time was used as a measure for testing the performance. The two datasets were synthetically generated, i.e., 100k_50 and 200K_50, and run on the Hadoop cluster with varying number of nodes. The algorithm was run on both datasets on 2, 3, 4 and 5 nodes in order to compare the decrease in time needed to run the algorithm. The results obtained are shown in Table III and Fig. 5.

Fig. 5 provides some really interesting insights into Hadoop and the algorithm in general. The running time on two nodes was used as a reference to compare the decrease in time when running the algorithm on the two datasets on different number of nodes. As shown in Fig. 5, as the size of the dataset increases, the time gained by running the algorithm on multiple nodes increases considerably. Consider the case of running the program on the 200K transactions dataset. The percentage decrease in time by running it on 5 nodes is close to 64% as compared to running it on 2 nodes. This is a significant increase in the efficiency of the algorithm.

TABLE III. RUNNING TIME ON DIFFERENT DATASETS

Nodes	Percentage drop in time	
	100K	200K
2	0	0
3	25.9	32.9
4	29.84	36.68
5	27.8	64.37

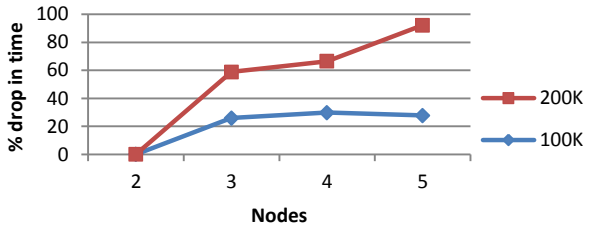

Figure 5. Comparison of running time on different datasets

Another interesting result observed is that the program scales well for larger datasets. As is evidenced, in almost every instance the time saved by running the larger dataset on multiple nodes is higher than the time saved by running the program on smaller dataset. This can be attributed to the fact that when running the algorithm in parallel on individual data blocks, the amount of time needed to perform the counting decreases. In Hadoop, we use HDFS as the distributed file system. HDFS splits the entire file into a number of blocks. So, if the number of nodes working on individual data blocks simultaneously is increased, the time is reduced. Thus, it is very important to choose the cluster size and file split size depending upon the data.

Another important observation relates to the 100K transaction. When running the algorithm on 4 nodes and then on 5 nodes, instead of time decrease which was expected, there was actually an increase in time. The job specifics were studied in order to understand this result and it was found that even though there were 5 nodes in the cluster, as a result of the division of the file, only 4 nodes were participating in the Map and Reduce phases. This unnecessarily caused some overhead in communication. So, it is very important to fix the file split size appropriately depending on the cluster. This is the same reason why there was a spike in the decrease in percentage of time as we added one more node to 4 nodes in the case of 200K transactions. The JobTracker was again studied to see what caused such a spike and noticed that when there were 5 nodes in the cluster, each Map task worked on an individual data block. A Map task is nothing more than a running instance of the Map class on the client node. Whereas in the case of 4 nodes, there were some pending map tasks which had to be computed again. Hence, it caused the steep increase in performance of the program. This reiterates how important the factors controlling the Hadoop environment are. There are many other factors to affect how Hadoop may

 ISBN: 978-1-61208-272-1

be configured, such as setting the number of map tasks, reduce tasks, defining our own split function, etc.

E. Analysis on Running Time vs. Transaction Length

In this analysis, the goal was to test how the running time varies based on the length of the transaction. The tests confirmed what was expected. A 200K transaction database generated synthetically was chosen. The average lengths of the transactions were chosen to be 20, 30, and 50. The results are shown in Table IV and Fig. 6.

TABLE IV. RUNNING TIME VS. TRANSACTION LENGTH

Data	Time(min)
200k_50	28
200k_30	16
200k_20	4.5

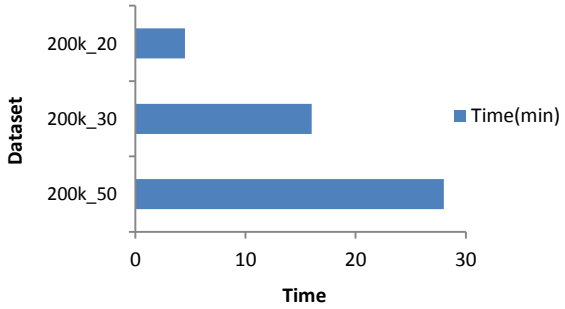

Figure 6. Running time vs. transaction length

The results for this analysis are straight-forward. When the transaction size is small, there is fast processing to count the candidate itemsets; hence, less time.

V. CONCLUSION

In this paper, the design, implementation, and evaluation of using Hadoop as a distributed framework for association rule mining was presented. This proves that a parallelized version of the Apriori could be very efficient and easy to port to Hadoop. The IBM smart cloud was also used and it shows the huge potential that cloud computing has to offer to organizations performing data mining tasks. The cloud provides an easy-to-use interface and web consoles to launch the cluster with pre-installed software and network connectivity. This is particularly useful as the programmer can instead concentrate on writing efficient algorithms and optimizing the data analysis rather than worrying about the maintenance of the Hadoop cluster.

For this paper, the main goal was not to optimize the Apriori algorithm but to test whether it can be implemented on Hadoop with satisfactory results. A modified version of the algorithm was designed and provided very good results with respect to the platform; yet it is currently slow. The speed of the algorithm may be improved to a large extent by using Tries and other alternative implementations for counting the candidates.

REFERENCE

[1] R. Agrawal and J. Shafer. “Parallel Mining of Association Rules,” IEEE Trans. Knowledge and Data Eng., Vol. 8, No. 6, 1996, pp. 962–969.
[2] R. Agrawal and R. Srikant, “Fast algorithms for Mining Association Rules,” Proc. Int’l Conf. Very Large Database, 1994, pp. 487-499.
[3] S. Brin, R. Motwani, J. D. Ullman, S. Tsur, “Dynamic Itemset Counting and Implication Rules for Market Basket Data,” Proc. ACM Conf. Management of Data, 1997, pp. 255-264.
[4] D. Cheung et al., “A Fast Distributed Algorithm for Mining Association Rules,” Proc. Int’l Conf. Parallel and Distributed Information Systems, 1996, pp. 31–42.
[5] D. Cheung and Y. Xiao, “Effect of Data Skewness in Parallel Mining of Association Rules,” Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, Vol. 1394, Springer-Verlag, 1998, pp. 48–60.
[6] F. Coenen and P. Leng, “Partitioning Strategies for Distributed Association Rule Mining,” The Knowledge Engineering Review, Vol. 21, No. 1, 2006, pp. 25 - 47.
[7] J. Cryans, S. Ratté, R. Champagne, “Adaptation of Apriori to MapReduce to Build a Warehouse of Relations between Named Entities across the Web,” Proc. Int’l Conf. Advances in Databases, Knowledge, and Data Applications, 2010, pp. 185-189.
[8] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” Communications of the ACM, Vol. 51, No. 1, 2008, pp. 107-113.
[9] E.H. Han, G. Karypis, V. Kumar, “Scalable Parallel Data Mining for Association Rules,” ACM Conf. Management of Data, 1997, pp. 277–288.
[10] J. Han, J. Pei, Y. Yin, “Mining Frequent Patterns without Candidate Generation,” Proc. Conf. Management of Data, 2000, pp. 1-12.
[11] L. Li and M. Zhang, “The Strategy of Mining Association Rule Based on Cloud Computing,” Proc. Int’l Conf. Business Computing and Global Informatization, 2011, pp. 475-478.
[12] J.S. Park, M. Chen, P. S. Yu, “An effective Hash-Based Algorithm for Mining Association Rules,” Proc. ACM Conf. Management of Data, 1995, pp. 175-186.
[13] J.S. Park, M. Chen, P.S. Yu, “Efficient Parallel Data Mining for Association Rules,” Proc. Int’l Conf. Information and Knowledge Management, 1995, pp. 31–36.
[14] A. Savasere, E. Omiecinski, S. Navathe, “An Efficient Algorithm for Mining Association Rules in Large Databases,” Proc. Int’l Conf. Very Large Database, 1995, pp. 432-444.
[15] T. Shintani and M. Kitsuregawa, “Hash Based Parallel Algorithms for Mining Association Rules,” Proc. Int’l Conf. Parallel and Distributed Information Systems, 1996, pp. 19–30.
[16] X. Yang, Z. Liu, Y. Fu, “MapReduce as a programming model for association rules algorithm on Hadoop,” Proc. Int’l Conf. Information Sciences and Interaction Sciences, 2010, pp. 99-102.
[17] M.J. Zaki, S. Parthasarathy, M. Ogihara , W. Li, “Parallel Algorithms for Fast Discovery of Association Rules,” Data Mining and Knowledge Discovery: An Int’l J., Vol. 1, No. 4, 1997, pp. 343-373.
[18] M. J. Zaki. Parallel and Distributed Association Rule Mining: A Survey. IEEE Concurrency, Vol. 7, No. 4, 1999, pp. 14-25.
[19] http://fimi.ua.ac.be
[20] http://hadoop.apache.org
[21] http://www.cs.umb.edu/~laur/ARtool/index.html

 ISBN: 978-1-61208-272-1