International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

An Efficient Parallel Association Rule Mining Algorithm based on Map Reduce Framework

Brijendra Singh[1]
[1]M.Tech Scholar, CSE
Dr. C. V. Raman University,
Kota, Bilaspur, Chhattisgarh, India

Rohit Miri[2]
[2]Assistant Professor, CSE
Dr. C. V. Raman University,
Kota, Bilaspur, Chhattisgarh, India

Abstract— **Data mining is an important field in Technology world. Association rules are a must and important step to discuss the data mining and inside findings of the relation between data variables of the database. In this Paper we have discussed an efficient parallel algorithm for association rules mining based on MapReduce framework. This can make performance of algorithm better and also reduce processing time.**

***Keywords*— Apriori, MapReduce, Data Mining, Hadoop, Parallel Association rule.**

I. INTRODUCTION

Data mining is hot topic in Internet world. It is also known as knowledge discovery in database [1]. To understand the data, we have to perform various steps in mining techniques [2] [18] like clustering, classification [19] and association rule mining in datasets from database. In this paper we focuses on association rule mining. Association rule mining [3] is one of the main techniques. It discovers the relation between data and projects the hidden information. These can be found by the finding valuable rules that are called the association rules [3,4].

With this valuable information, data mining attracts every area like researchers, managers, industries, healthcare, scientists etc. it may helps in decision making, prediction, pattern learning. Association rule mining covers various areas but since its inception it is used in the market basket [5] analysis to find the products that were sold together frequently. This gives an idea to companies to arrangement of their products and making the decisions based on that.

Here are some of the examples of benefits of association rules mining.

- Crime detection and prediction: frequent pattern analysis in the criminal database helps in finding the city areas and crimes that has been repeatedly happens [6,7].
- Cyber security: frequent pattern analysis of log files of networks and tracking details, patterns of suspicious IP address and ports to prevent the attacks [8].
- **Crowd mining**: finding information from social data to achieve better behavior of the residents [9,10].

And many fields can be benefitted by the association rule mining.

Apriori [11], Eclat [12] and FP-Growth [13] are some important association rule mining algorithms. Most of algorithms scans the datasets and find frequent pattern, latter generate the association rules. The Apriori algorithm given by R. Agrawal [11] works on the concept if an itemset is frequent then its non-empty subsets must be frequent. It performs the iteration in which result of previous step will become input of the next step to find the frequent itemsets. First step is to find the singleton frequent itemset that occur more times than minimum user defined value (minimum support). Now k-frequent itemsets found by k-1 frequent itemsets based on Apriori. So on for k+1 frequent itemsets we have to perform k^{th} iteration and this iteration performs until all frequent itemsets not found.

Apriori algorithm scans the whole database at each pass, this will results in higher I/O cost to system. But with rapid growth of the Internet and IOT, data is expanding. Database becomes larger and larger to manage by the traditional Apriori algorithm.

Parallel association rule mining algorithms are needed to solve above problem. R. Agrawal and John C. Shafer [14] presented three algorithms for parallel association mining rules. These rules were based on the Apriori algorithm. We can say it was algorithms to run Apriori algorithm in parallel computing environment. These are CD (Count Distribution), DD (Data Distribution), CaD (Candidate Distribution). However due to weakness of synchronization and communication, these algorithms are not capable of solving the parallel association rule mining. MapReduce [15] is a framework that easily implements programming task in large data. In the context of MapReduce this large data is called as Big Data or large Data cluster. Using this framework it is become easy to implement parallel association rule mining. This framework is presented by the Google [16]. MapReduce [17] programming framework has fault tolerance, handles failures and provides good working environment to the programmers. It is an open source framework. Programmers easily implement their work in MapReduce Framework.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

II. BACKGROUND

Association Rules

Let I = {I_1, I_2, I_3,…..,I_m} are the set of items and m is number of items. Transaction database is DB = {T_1, T_2, T_3,…..,T_n} and $T_i \subseteq I$. We can say itemset X contained in T_i only if $X \subseteq T_i$.

Association rule X=>Y says that $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \varnothing$. Two main key points that tells association rule are support and confidence [20].

Sup(X) = frequency of X in the database and

Conf(X=>Y) = sup(X U Y)/sup(X).

Association rules [4, 20] mining algorithms finds frequent itemsets by setting minimum support and confidence for itemsets. These values set by the users and may vary for different applications.

Apriori Algorithm

Apriori [11] is an important algorithm for selecting frequent itemsets using candidate generation. This is also known as fasted algorithm proposed.

1. Search all elements (1-element itemsets) one by one, so that they have minimum support. Here we denote support by s.
2. Repeat
    a. By the previous result of i-itemsets, search for the i+1 itemsets which have minimum support.
    b. Now this will be i+1 itemsets that are frequent.
3. Do till, itemsets size reached to maximum.

MapReduce

MapReduce is introduced by the Google [16] and under the MapReduce framework we can easily implements parallel, distributed algorithms. Google given it to Apache Software Foundation [22], now it is open source and developed under Apache Software Foundation. It is part of Hadoop [23,24] that can handle BIG Data [21] and large Applications. MapReduce [25,26] contains two components. One is map(), its work is filtering and sorting and other is reduce(), its work is summary operation like counting work. Input data is divided into different portion and then it send to mapper, will do the filtering and sorting arrange data in (key, value) pair then it send to reducer. Reducer runs reduce function and calculate the output. Both map and reduce function written by the programmer as per their task.

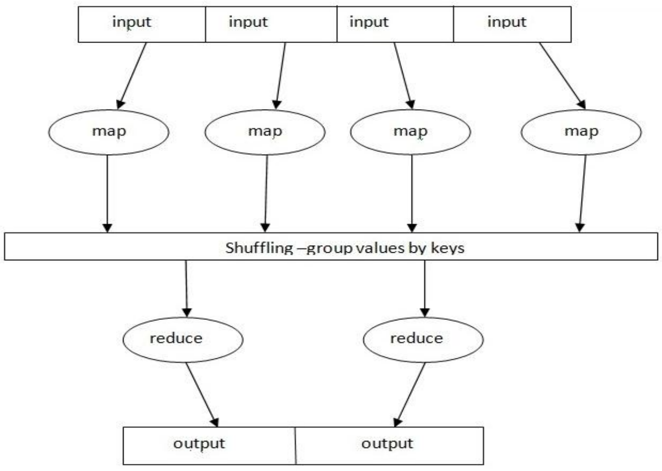

Figure 2.1 MapReduce Working Explanations

These functions will be represented in this way:

Map: (key1, value1) => list (key2, value2)

Reduce: (key2, list (value2)) => (key3, value3)

III. RELATED WORK

Apriori algorithm cannot handle large amount of data. As the size of data increases I/O cost in Apriori algorithm increases. So Apriori algorithm was implemented in parallel way by R. Agrawal [14], but due to synchronization and communication problem performance is not good in case of large amount of data or when we talk of BIG DATA. Then Apriori Algorithm is implemented in MapReduce Framework [27]. There are two steps in Apriori algorithm, one is candidate generation that finds the frequent itemsets and add them to candidate sets. Second is count step, in this step all candidate itemset compared with minimum support then subsets which fulfil the criteria (minimum support) can be selected as frequent itemsets. Mapper performed at first step by dividing datasets into key value pair, find the potential candidate set. Then reducer do the reducing part here which set qualify minimum support such candidate will be selected as frequent item sets.

In traditional Apriori algorithm there are two step in which association rule mining is performed. First step is candidate generation step; in this step all frequent itemsets will be generated from previous pass generated itemsets. This process performed in iterative manner in k-pass that gives k-frequent itemsets. Second step is count step, in this step all candidate itemsets prune to frequent itemsets which have minimum support defined by the user.

In implementation of parallel Apriori algorithm in mapreduce framework [28,29] author follows two steps and implemented it. This shows good performance as compared to the traditional Apriori. But various implementation of Apriori algorithm in serial or in parallel shows that for 2nd iteration, time taken by the algorithm is much higher than any other pass. Another paper [30] we studied that implements different data structures in the mapreduce framework for Apriori algorithm. In our implementation we try to reduce the time taken by 2nd iteration so that our proposed algorithm enhanced the performance of Parallel Apriori Algorithm in MapReduce Framework.

IV. PROPOSED PARALLEL APRIORI ALGORITHM BASED ON MAPREDUCE

Our proposed algorithm consists of three phases.

A. Phase 1:

The transactional database from system loaded to HDFS file system. This database of HDFS file system divided into blocks, by default of 64MB and putted at data nodes. Data nodes range from one to many. Now Map() function takes input of data in key / value pair form, where key is the offset in bytes of this record to the start point of the data file and value is a string of the content of this record. Map() function perform on this data pair and convert into (item, 1) key / value pairs. Now from different mapper output will merge and sorted in fashion such that similar item comes in one area. This is done by framework itself we need not to worry about that. Now Reduce() function is invoked to find out the frequency of item and prune which item having less frequency than minimum support defined by user. Remaining items stored in singleton-frequent itemset.

Pseudo code for Phase 1:

```
a. Foreach transaction T in $R_i$
b. Map(line offset, T)
c. Foreach item in T
d. Yield(I,1)
e. End Foreach
f. End Map()
g. ReduceByKey(I,count)
h. Sum=0
i. While(item I in participation)
j. Sum+= count
k. End while
l. If(sum >= min_sup)
m. Yield(I,sum)
n. End if
o. End ReduceByKey
```

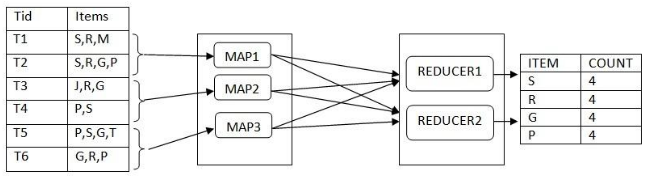

Figure 4.1 Graphical Diagram for Phase-1

B. Phase 2:

In the second phase we store the singleton-itemset in a bloom filter. Bloom filter [31] works on item of length 1. It can easily store and determine the membership of item. Mappers take every transaction and prune it so that it contains items which presented in bloom filter and produce all possible pruned transaction. Reducer does same task and calculate total count of each pair. Which pairs having total count more than minimum support will be selected as frequent itemsets. The result of mapper and reducer are similar as mapreduce classic implementation but this approach takes less time.

Pseudo code for Phase 2:

```
a. If number of singleton frequent itemsets are large
b. Foreach transaction T in $R_i$
c. $F_T$ = intersect(T,$L_{K-1}$)
d. $C_T$ = pair $F_T$
e. Foreach candidate C in $C_T$
f. Yield(C,1)
g. End Foreach
h. End Foreach
i. ReduceByKey(I, count)
j. Sum=0
k. While(item I in partition)
l. Sum+= count
m. End while
n. If(sum>=min_sup)
o. Yield(I,sum)
p. End if
q. End ReduceByKey
r. Else
s. Iterate phase 3 for n= 2 to m
t. End else
```

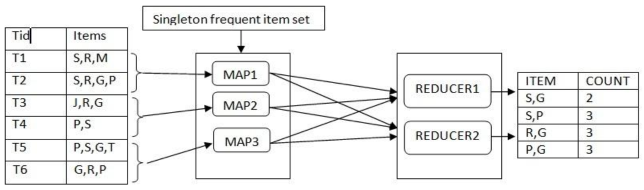

Figure 4.2 Graphical Diagram for phase-2

C. Phase 3:

Here we use same approach as phase 1. We take k-frequent itemset and by them generate (k+1) frequent itemsets. Both map() function and reduce() function works same like phase 1. At the first iteration input will be 2-frequent itemsets output of phase 2. We perform this iteration till $C_{ki}>1$ where C_{ki} is candidate set of k size and ith iteration.

Pseudo code for Phase 3:

```
a. Read $L_{k-1}$ from HDFS
b. $Can_k$ = ap_gen($L_{k-1}$)
c. Foreach Transaction T in $R_i$
d. Map(line offset, T)
e. $C_T$ = subset($Can_k$,T)
f. Foreach candidate c in $C_T$
g. Yield(C,1)
h. End Foreach
i. End Map()
j. End Foreach
k. ReduceByKey(I,count)
l. Sum=0
m. While(item I in participation)
n. Sum+= count
o. End while
p. If(sum >= min_sup)
q. Yield(I,sum)
r. End if
s. End ReduceByKey
```

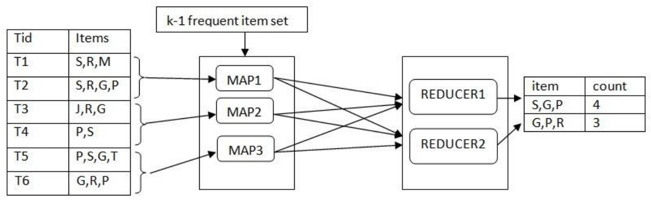

Figure 4.3 Graphical Diagram for phase-3

V. EXPERIMENTAL RESULTS

In this section, we evaluate performance of our proposed algorithm. Here we are using ARtool [32] to generate synthetic data. ARtool is an open source tool package. This tool is developed by university of Massachusetts and it is provided to dataset generation and other tasks.

We used Hadoop version 1.2.1, one machine for name node and 2 machines for data nodes. Each machine is having core i5 intel processor, 2 GB of RAM and 500 GB of storage. Every machine is having ubuntu 12.04 LTS 64 bit installed and setup with jdk 1.7.0_79 and jre 1.7.0_79.

A. Datasets:

Experiments were done with datasets generated by ARtool, this open source tool can generate synthetic data. We used dataset T1000_AT10_I100_P50_AP5, this database contains 10^6 transactions and 870 number of items in it. We choose this tool because mostly studies used IBM data Generator and which is obsolete now.

B. Performance Analysis:

Performance of our proposed algorithm is evaluated with the datasets. Its result may be somewhat differ from other authors because here we used machine, which are really low end in terms of specification. But it is perform very well in large data. Our algorithm outer perform for the 2nd iteration which hugely reduces the time of 2nd iteration. Our result can be understood by performance graph given below in Figure 5.1.

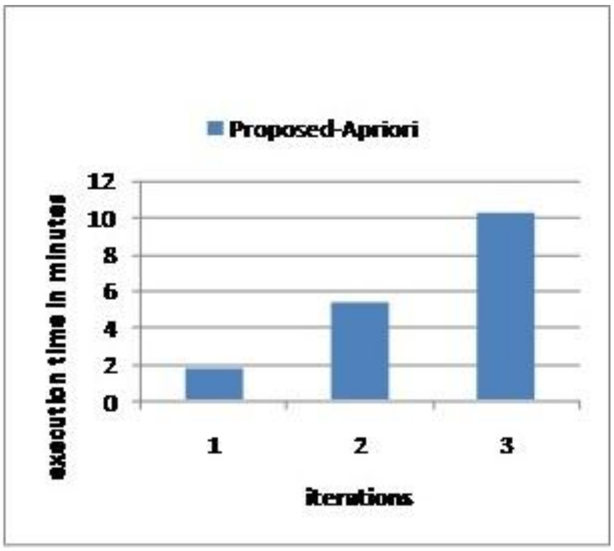

Figure 5.1 T1000_AT10_I100_P50_AP5 with min_sup 0.15%

We can also compare algorithm based on Apriori Algorithm with different implementation. It is plotted below in Figure 5.2.

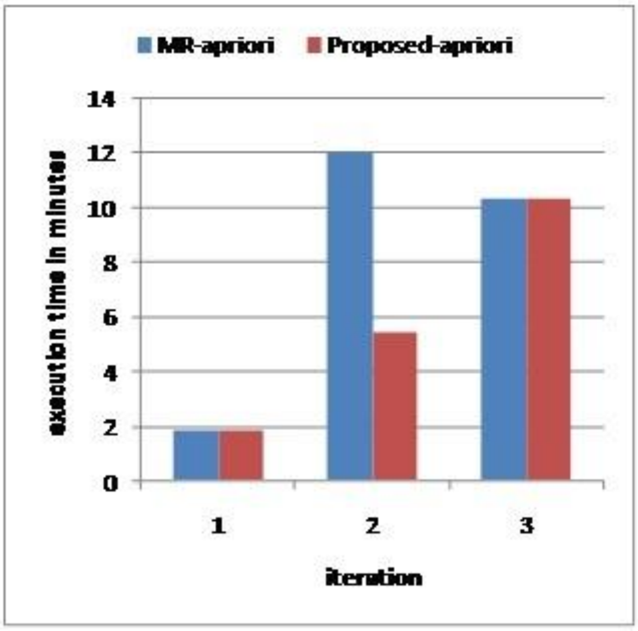

Figure 5.2 Comparison between MR-Apriori and proposed-Apriori

VI. CONCLUSIONS

In our Paper, we have proposed an algorithm based on Apriori algorithm in parallel implementation based on MapReduce Framework that can improve processing time for the second iteration in frequent itemsets mining. Our algorithm easily handles large amount of data for mining with less processing time. We can also make improvement in algorithm by clean code or changes in count step. Other programming framework also now developed that can work faster than MapReduce Framework. In future we try to implement it with more efficient Framework. For that we have further study on the subject.

ACKNOWLEDGMENT

I would like to express our sincere thankfulness to my guide for their helpful support and valuable advice during our research work.

REFERENCES

[1] https://en.wikipedia.org/wiki/Knowledge_Discovery_in_Databases.
[2] J. han, M. Kamber and J. Pei, “Data Mining: Concept and Techniques” 3rd ed. San Francisco, 2011.
[3] Agrawal R, Imielinski T and Swami A., “Mining association rules between sets of items in large database”. In Proc. Of the ACM SIGMOD conference on Management of Data, pp. 207-216, Washington, D.C., May 1993.
[4] Osmar R. Zaiane, Mohammad El-Hajj, Paul Lu. Fast ParallelAssociation Rule Mining Without Candidacy Generation, Technique Report.
[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen & A.I. Verkamo, 1996. Fast discovery of association rules. Advances in Knowledge Discovery and Data Mining (U. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy, ed.), American Association for Artificial Intelligence.

[6] Anna L. Buczak and Christopher M. Gifford. Fuzzy ssociation Rule Mining for Community Crime Pattern Discovery. In *ISI-KDD* 2010, ACM, USA, 2010.
[7] Tong Wang , Cynthia Rudin, Daniel Wagner and Rich Sevieri. Learning to Detect Patterns of Crime. In *Springer*, MIT, USA, 2013.
[8] Latifur Khan, Mamoun Awad and Bhavani Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. In *VLDB Journal*2007, pp: 507-521, 2007.
[9] Yeal Amsterdamer, Yeal Grossman, Tova Milo and Pierre Senellart. Crowd Mining. In *SIGMOD'13*, USA, 2013.
[10] Yeal Amsterdamer, Yeal Grossman, Tova Milo and Pierre Senellart. CrowdMiner: Mining association Rules from the crowd.In *Proceedings of VLDB Endowment*, 2013.
[11] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proc. 1994 Int. Conf. VeryLarge Data Bases, pages 487-499, Santiago, Chile, September 1994.
[12] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara and Wei Li. *New algorithms for fast discovery of association rules*. Technical Report 651, Computer Science Department, University of Rochester, Rochester, NY 14627. 1997.
[13] J. Han, H. Pei and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX)*, ACM Press, New York, NY, USA 2000.
[14] Agrawal, R., & Shafer, J. C. (1996)." Parallel mining of association rules". Knowledge and Data Engineering, IEEE Transactions on, 8(6), pp. 962- 969.
[15] https://en.wikipedia.org/wiki/MapReduce
[16] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp. 107-113.
[17] Yang, X. Y., Liu, Z., & Fu, Y. (2010, June). MapReduce as a programming model for association rules algorithm on Hadoop. In Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on (pp. 99-102). IEEE.
[18] S Pandey, R Miri, SR Tandan (2013) "Diagnosis And Classification Of Hypothyroid Disease Using Data Mining Techniques " International Journal of Engineering Research and Technology (IJERT).
[19] MK Shrivastava, P Chouksey, R Miri (2013) "Exploring Data Mining Classification Techniques"International Journal of Engineering Research and Technology (IJERT).
[20] https://en.wikipedia.org/wiki/Association_rule_learning
[21] Gordon, K. (2013). What is Big Data?. ITNOW, 55(3), pp. 12-13.
[22] https://en.wikipedia.org/wiki/Apache_Software_Foundation
[23] Apache hadoop. *http://hadoop.apache.org/*2013.
[24] https://developer.yahoo.com/hadoop/tutorial/
[25] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), pp.107-113.
[26] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., & Stoica, I. (2008, December). Improving MapReduce Performance in Heterogeneous Environments. In OSDI (Vol. 8, No. 4, p. 7).
[27] Li N., Zeng L., He Q. & Shi Z. Parallel Implementation of Apriori Algorithm Based on MapReduce. In *Proc. of the* 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD '12), Kyoto,IEEE: 236 – 241, 2012.
[28] J. Dean and S. Ghemawat. MapReduce: Simplified dataprocessing on large clusters. In *Proc. OSDI. USENIX Association*, 2004.
[29] Lin M., Lee P. & Hsueh S. Apriori-based Frequent Itemset Mining Algorithms on MapReduce. In *Proc. of the 16th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12)*, New York, NY, USA, ACM: Article No. 76, 2012.
[30] Sudhakar S, Rakhi G, P.K. Mishra "Performance analysis of Apriori algorithm with different data structures on hadoop cludter" October 2015, International Journal of computer applications (IJCA).
[31] https://en.wikipedia.org/wiki/Bloom_filter
[32] http://www.cs.umb.edu/~laur/ARtool/index.html