A Novel Approach of Frequent Itemset Mining Using HDFS Framework

Prajakta G. Kulkarni and S. R. Khonde

Abstract Frequent itemset extraction is a very important task in data mining applications. This is useful in applications like Association rule mining and co-relations. They are using some algorithms to extract the frequent itemsets, like Apriori and FP-Growth. The algorithms used by these applications are inefficient to support balancing, distributing the load, and automatic parallelization with good speed. Data partitioning and fault tolerance is also not possible because of excessive data. Hence, there is a need to develop algorithms which will remove these issues. Here, a novel approach is used to work on the extracting the frequent itemsets using MapReduce. This system is based on the Modified Apriori, called as Frequent Itemset Mining using Modified Apriori(FIMMA). To automate the data parallelization, well balance the load and to reduce the execution time FIMMA works concurrently and independently using three mappers. It uses decomposing strategy to work concurrently.

Keywords Association rules · Frequent itemsets · Data partitioning
Load balancing · MapReduce · Hadoop · FIMMA

1 Introduction

Frequent itemset extraction is the basic problem in data mining applications, such as association rule, correlations, sequences, and many more data mining tasks. Hence, this becomes an important research topic to extract the frequently used itemsets. These frequent patterns are useful to take decisions in product marketing, sales, etc. [1]. Association rule mining is popular in data mining [2]. The main goal of Association rule is to find all the rules that fulfill a user-defined threshold. The first

P. G. Kulkarni (✉) · S. R. Khonde
Modern Education Society's College of Engineering, Pune 411001, India
e-mail: prajakta.r999@gmail.com

S. R. Khonde
e-mail: shraddha.khonde@mescoepune.org

S. Bhalla et al. (eds.), *Intelligent Computing and Information and Communication*,
Advances in Intelligent Systems and Computing 673,
https://doi.org/10.1007/978-981-10-7245-1_43

phase of association rule is to identify frequent itemsets whose support is greater than the threshold and the second phase is to form conditional implication rules, among the frequent itemsets. Frequent itemsets generation defines the two similar itemsets. The first itemset is similar to another. Now a day, there are enormous data generated from different areas such as IT companies and web applications. Existing data mining applications are unable to handle vast data and are only suited for a typical database. Thus, to extract the frequent itemsets from the excessive database is a very critical task [3]. For better utilization of frequent itemsets using large size database, speed is very important. Speeding up the process of FIM is very complex because it consumes most of the time to calculate the input/output intensity. In this modern era, datasets are excessively large and sequential FIM algorithms are unable to compute large database. They, however, failed to analyze data accurately and they suffer from performance degradation. To solve these problems, MapReduce is used to calculate frequent itemsets. Using this approach, the data will not only be distributed in an efficient way but also balanced in the cluster. Hence, the performance of finding frequent itemsets will be optimized [3].

This MapReduce is using the FIM which is based on the Modified Apriori, called FIMMA. In this strategy, we are focusing the data partitioning method, load balancing of data with a parallel approach. FIMMA consumes less time compared with the traditional Apriori. The working of mappers and reducers is done concurrently to optimize the speed, well balancing the load across various clusters [3].

The rest of this paper is partitioned as follows. Section 2 gives the review of the literature. Section 3 defines the problem statement. Section 4 gives the present system architecture. Section 5 explains the algorithms and methodology for the system and discussed the expected results in Section 6. Section 7 concludes this paper.

2 Related Work

The authors of "Association Rule mining extracting frequent itemsets from the large database" have presented a problem of finding the frequent items from the excessive database. The authors have developed the rules that have minimum transactional support and minimum confidence. For this, an algorithm is used that carefully estimates the itemsets for one pass. It adjusts the data between the number of passes and itemsets that are measured in a pass. This process uses pruning system for avoiding certain itemsets. Hence, this gives exact frequent itemsets from excessive databases [4, 5]. A number of parallelization procedures is used to increase the performance of Apriori-like algorithms to find frequent itemsets. MapReduce has not only created but also exceeds in the mining of datasets of gigabyte scale or greater in either homogeneous or heterogeneous groups. The authors have implemented three algorithms, DPC, FPC, and SPC [6]. SPC has straightforward functions and the FPC has static passes merged checking capacities. DPC consolidates the dataset of various lengths by utilizing dynamic strategy and it gives good

performance over the other two calculations. Accordingly, these three calculations will scale up to the expanded dataset [6].

When dataset gets larger the mining algorithms becomes inefficient to deal with such excessive databases. The authors have presented a balanced parallel FP-Growth algorithm BPFP [7], a revised version of PFP algorithm [4]. FP-growth algorithm is utilized with the MapReduce approach named as Parallel FP-growth algorithm. BPFP balances the load in PFP, which boosts parallelization and automatically enhances execution. BPFP gives a good performance by utilizing PFP's grouping system [7].

FIUT suggests a new technique for mining frequent itemsets called as Frequent Itemset Ultrametric Tree(FIUT) [8]. It is a sequential algorithm. It consist two main stages to scan the database. First-stage calculates the support count for all itemsets in a large database. The second stage uses pruning method and gives only frequent itemsets. While calculating frequent one itemsets, stage two will construct small ultrametric trees. These results will be shown by constructing small ultrametric trees [8]. Dist-Eclat, BigFIM are two FIM algorithms used with MapReduce Framework. Dist-Eclat focuses on speed by load balancing procedure using k-FIS. BigFIM concentrates on hybrid approach for mining excessive data [9]. Apriori algorithm is additionally used to create *k*th FIS itemsets. The *K*th FIS is used to search frequent itemsets based on the Eclat system. These three algorithms are used with round-robin technique which achieves a better data distribution [9].

PARMA uses parallel mining approach with the benefits of Randomization for extracting frequent itemsets from a vast number of databases. This divides the functionality into two parts, gathering the arbitrary data samples and secondly it uses parallel computing method that is utilized to increase the mining speed. This method avoids the replication that is very expensive. A mining algorithm applies to every segment individually with parallel approach [10]. K-Nearest Neighbor Joins utilizes MapReduce and distributes the excessive information on the number of machines. This is done by the mappers and the reducers give the results in terms of the KNN join. KNN Join is the key component to search the *k*th-nearest neighbor. MapReduce is utilized for effective computing the data to obtain the best performance result [11, 12]. To diagnose the Heterogeneous Hadoop Cluster and to search primary faults, this paper is used Hadoop schedulers to produce efficient Hadoop clusters even if they are in heterogeneous clusters [13]. It proposes the DHP algorithm (direct hashing and pruning) which is used for minimized candidate set generation for large itemsets. It solves performance degradation problem of large dataset mining. It minimizes candidate itemsets.

FIUT is used with MapReduce to find frequent itemsets. MapReduce is a popular programming approach used for computing massive datasets [14]. It divides into three MapReduce phases. The database is divided into number of input files and given to each mapper. The first MapReduce phase finds out frequent-1 itemset and Second MapReduce phase scans the frequent one itemsets and generates k-frequent itemsets. Third MapReduce phase uses FIUT algorithm and it will create ultrametric tree [3]. FiDoop-DP Data Partitioning uses the Map Reduce programming and gives the effective data partitioning technique for frequent

itemset mining. This increases the performance by using the data partitioning technique, which is based on the Voronoi diagram. It extracts the correlations between the transactions. By consolidating the similarity and the Locality-Sensitive Hashing strategy, FiDoop-DP puts most similar records in data partition to increase locality and this is done without repeating records [5, 15]. To differentiate and extract frequent and infrequent itemsets from the massive database two-phase scanning will be done here. In the first scan, it accepts input and distributes it into mappers and finds out infrequent itemsets using minimum support. The reducer combines the result and sends it to the second phase. In this phase, it scans first phase output and gives the final result [16].

3 Problem Statement

Problem statement concentrates on the investigation of Frequent Itemset Ultrametric Tree (FIUT) and to find the efficient way for its execution in HDFS framework Implementation on FIMMA. To show that the proposed algorithm on the cluster is sensitive to data distribution and dimensions, as itemsets with different lengths have different decomposition and construction costs. Improving energy efficiency of FIMMA running on the Hadoop clusters. To improve the performance, a workload balance metric to measure across the clusters computing nodes is developed.

4 System Architecture

FIMMA suggests parallel frequent itemset mining algorithm which uses MapReduce programming technique for development. This removes the issues of existing system and applies automatic parallelization, balancing the load of the excessive database, and well distribution of given data. FIMMA is based on Modified Apriori algorithm to overcome the issues of FIUT algorithm with reduced time. It uses hash-based technique [17].

4.1 Objectives of Proposed System

- Better performance and improved accuracy using automatic parallel processing. It performs with less time execution to scan the database.
- Keeping the cost constraint as it is and dealing with load balancing with automatic parallelization.
- Hashing technique is used to differentiate the traditional Apriori.

The proposed system consists three MapReduce phases. The user is responsible to give the input. This input will be accepted by the job manager as shown in Fig. 1. Job manager splits the input into a number of blocks, processes it and gives the output to the reducer. This output is in the form of frequent one itemset.

The output of the first MapReduce is applied to the mapper of the second phase. It scans all the data to give frequent-k itemsets. It uses pruning method to find out frequent and infrequent itemsets. The result is obtained from reducer, in the form of k-frequent itemsets. The system architecture is as shown in Fig. 1. This is distributed into two main parts.

1. HDFS framework
2. MapReduce Approach.

HDFS is a Hadoop distributed file system and used to store the log files. HDFS framework accepts the data from the user. The user gives the input using SQL queries or in the form structured data and uploads it. This uploaded dataset accessed by the Job manager. Job manager is responsible to distribute the dataset to the available mappers of each data node.

Here, the transactions from input dataset are distributed to the mappers. Each mapper access the input scans it and generates the results in terms of key and values. Key is the item-name and value is the item-count. Using the minimum support the mappers gives the result to the reducers in the form of key and values. Each mapper calculates frequent-1 itemset. Reducer combines the result of each

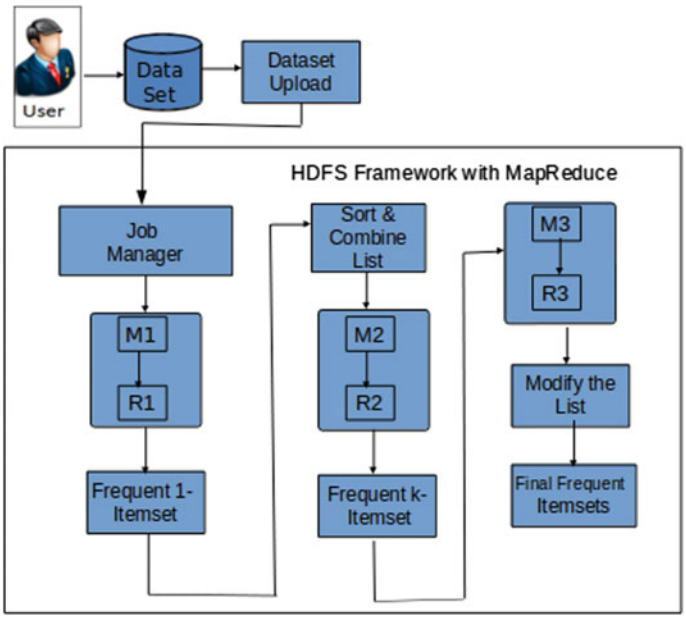

Fig. 1 System architecture

mapper, sorts it and generate a final list. This is frequent-1 itemset list. This output gives to the next MapReduce. It applies another round of scan. In this stage, it accepts the frequent-1 items and compares it with the minimum support and removes the infrequent items. This is called pruning system. Depending upon the users given threshold value of k this MapReduce generates k-frequent itemsets. (where k-itemsets < n number of dataset) It makes the possible combination of frequent itemset from each mapper and gives to the reducer. Second MapReduce updates the list and gives k-frequent itemset. This frequent-k-itemsets applies to the third MapReduce. In this MapReduce, it accepts all k- itemsets and gives the result in terms of top most k-frequent itemset.

FIMMA uses hash-based technique. There is a hash table used after each result of MapReduce. These hash tables are used to store the result generated from frequent-1 itemset to the k-itemset. These hash tables give a unique value to the stored frequent items. Unique value is obtained by calculating mod hash formula. Whenever there comes new input then hash table compares that items with stored one. When new input from first mappers matches with stored one then hash table sets a bit to 1 otherwise 0. This same procedure applies to the result of second MapReduce (i.e., k-frequent itemsets) and on the third MapReduce. In last MapReduce, it checks all set bit and stores into a new list, update it and gives the final result. FIMMA helps to reduce the candidate generation of items by using hash tables, therefore automatically it reduces the time. It controls the huge generation of candidates with minimum support. It avoids the transaction record which does not have any frequent items by comparing hash tables.

5 System Analysis

5.1 Algorithm for Frequent 1 Itemsets [3]

Input: minimum-support, Database D
Output: Frequent-1 itemset
Mapper Algorithm
Step 1: Mapper function is used with-MAPPPER (key offset, values Database D)
Step 2: //TR shows the transaction in Database D
Step 3: for loop is used for all Transactions TR in Database D do
Step 4: Candidate-items < – Splited each transaction TR.
Step 5: Use for loop for all candidate-items in all items, do
Step 6: output(candidate-item, 1)
Step 7: Here ends second for loop
Step 8: Here ends first for loop
Step 9: Mapper Function ends with each items count.

Step 10: The reducer takes input from mappers as input = (candidate-item, 1)
Reducer Algorithm
Step 11: REDUCE function starts with key, value (key candidate-item, values 1)
Step 12: take a variable total to store output, i.e., total = 0;
Step 13: Use for loop to calculate all candidate-item do
Step 14: Add new frequent item in total +=1 // Here ends for loop
Step 15: Output (frequent1-itemset, total).

5.2 Algorithm for Frequent-K-Itemsets

Input: minimum-support, Database D
output: frequent-k-itemsets.
Mapper Algorithm:
Step 1: In step 1 mapper function is used i.e. MAPPER (key offset, values Database)
Step 2: //TR shows the transaction in Database D
Step 3: for loop is used for all Transactions TR in Database D do
Step 4: Candidate-items ← Splited each transaction TR. //Here input database D is frequent-1 itemset
Step 5: Use for loop for all candidate-items in all item, do
Step 6: Step 6 applies pruning system using if condition for infrequent items
if Candidate-item = infrequent item then
Step 7: Remove the Candidate-item which is infrequent from the Transaction TR;
Step 8: If conditions ends here
Step 9: variable Frequent-k-itemset is used to store all k-frequent items and shown by-Frequent-k-itemset ← (frequent-k, fr-set)
//After applying pruning system fr-set is the result with frequent-k items
Step 10: output (Frequent– itemset, 1);
Step 11: Here ends second for loop
Step 12: Here ends first for loop
Step 13: Mapper function is ends here by giving all mappers output.
Reducer Algorithm
Step 14: Reducer starts from this step using function REDUCER (key k-itemset, values 1)
Step 15: Total = 0;
Step 16: used for a loop to count all items from mapper as-for all (k-itemset): do
Step 17: Total += 1;//For loop ends here
Step 18: output = (frequent-k items + total)
Step 19: Reducer function ends here with final k output.

5.3 Algorithm for FIMMA

Step 1: Consider C be the variable for selection of one cluster at a time
Step 2: Here, the database will be scanned using minimum support and it will generate frequent items. It will combine all the possible combinations of frequent itemsets
Step 3: Function Fre1 stores the frequent itemsets → Fre1 = find-freq-1itemset(T)
Step 4: for k = 2 to $f_{k-1} \neq \emptyset$; generate Ø from f_{k-1} items
Step 5: Consider H1 is the hash table of size 8. B1 is buckets in the hash table and A1 is Unique value to the frequent itemsets. V1 is the bit vector
Step 6: Calculate the items I up to user threshold value w from ck with min support; ft($1 \leq w \leq k$); end for
Step 7: Calculate frequent items with minimum support
Step 8: y variable to store the result of minimum support, i.e., y = min support (c_k, f_t)
Step 9: get transaction id in variable → target = get-trans id (y)
Step 10: compare the target values with the hash tables
Step 11: for each transaction id in target increment count of all ck
Step 12: if b1 >=min support; then bit vector v1 = 1; otherwise v1 = 0
Step 13: prune the 1 = 0 itemsets and modify the list
Step 14: f_k = items in c_k (min support)\\ end for

5.4 Mathematical Model

Let S be the system which do analysis and read documents; such that:

S = {S1, S2, S3, S4, S5} where—S1 represents a query requesting by the user; S2 represents authentication; S3 represents MapReduce module; S4 denote the sql injection techniques; S5 gives the graphical presentation.

S1 = {U1, U2, U3. . .. .Un}; Where, S1 contains SQL query–If S1 is valid then proceed, Else discard.

S2 define user is authenticated or not; Where, Ui = {UI1, UI2.UI3. . .. . .UIn}

- Ui is the master node which having different storage nodes as clusters.

S3 = Functionality for three MapReduce phases:

- Input: database DI,min sup; Output: Frequent itemsets
- Let DI = {DI1, DI2, ...Din}; where, DI is Input Database

Applying algorithms to Find 1 and k-frequent itemsets, Hash-based Algorithm
F1(freq 1-itemset) = scan $\sum 0^{DI}$
; Fk (freq k-itemset) = scan $\sum$ (F1)

$$FI = \{FI1, FI2, \ldots .FIK\}; \text{Where FI is the final frequent itemsets}$$
$$S4 = \{\text{patten } 1, \text{pattern } 2. \ldots \ldots \text{pattern n}\}$$

Each pattern checks the behavior of query created by end user's S1 (query module). $S1 = \text{avg} + \text{min} + \text{max} + \text{round} + \text{floor} + \text{todate} \,//\, \text{possible queries}$

$$S5 = \text{graphical representation for time comparison graph}$$

6 Experimental Results

The experimental results evaluated with the minimum support (sometimes larger data sizes). This upgradation comes at no execution cost, as they prove the way that this implementation, achieves the good performance, compared to other techniques with reduced time. By examining with this work, it demonstrates that, the execution of FIUT is slow. It results that, whenever increased the minimum support and dataset, it gives the balanced output. FIMMA gives an improved performance by using hash table concept. When the threshold value decreases, other methods occupy more memory as well as consume more time.

Table 1 shows the time required to extract frequent itemsets. As shown in table I, first column shows the Size of Dataset. Other columns show the methodologies to find frequent itemset. Proposed system requires less time compared to existing systems. It shows the time in seconds. The performance of this system against the Frequent Itemset Ultrametric Tree (FIUT) method is as shown in Fig. 2. Modified Apriori algorithm is a good algorithm to give the correct results as compared to existing systems. Also, this algorithm shows the faster execution even for a large database. As synthetic dataset is used here, the user can make their own dataset to run the tests.

Table 1 Time required for finding the frequent itemset from dataset

Size of dataset	FIUT	Modified apriori (FIMMA)
1000	139	113
2000	165	144
3000	218	196
5000	277	234
10,000	317	293

Figure 2 shows the graphical representation of the methodologies and datasets. These are the results of two methodologies when the dataset is increased. X-axis shows the methodologies and y-axis shows the time in seconds. Figures 3 and 4 shows the Hadoop implementation results. Figure 3 shows the FIUT implementation with required time in ms. In this, it takes 43,780 ms (shown in red color rectangle) to execute a job. Figure 4 shows the FIMMA implementation. Here it shows the time required to execute the same job. FIMMA takes 21,090 ms to execute a job. Hence, it shows that FIUT takes more time than the FIMMA method. FIMMA gives better performance than the FIUT with reduced time.

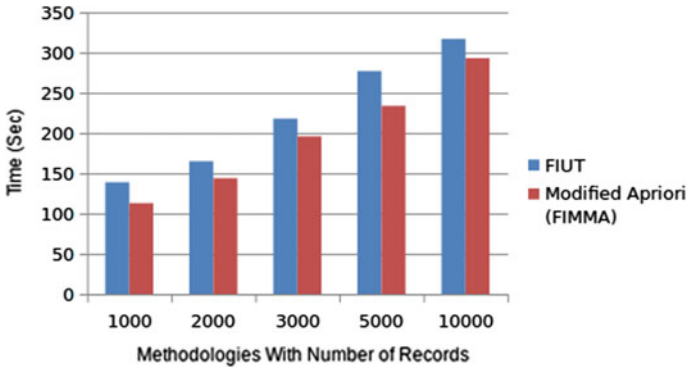

Fig. 2 Comparison graph

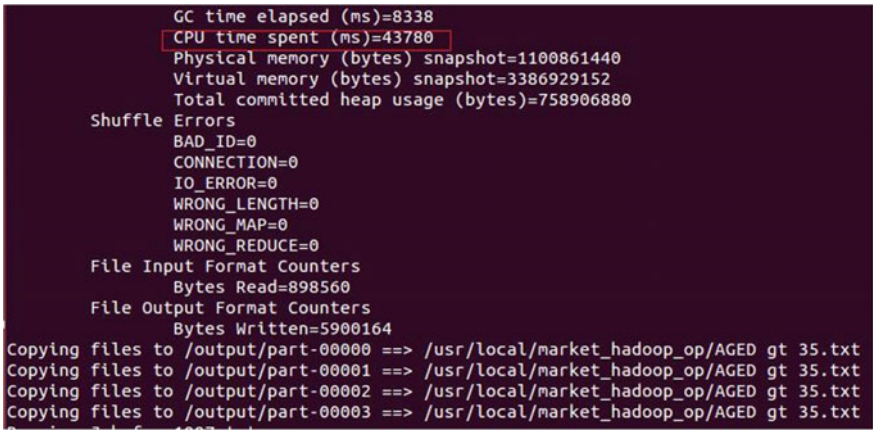

Fig. 3 Hadoop implementation using FIUT

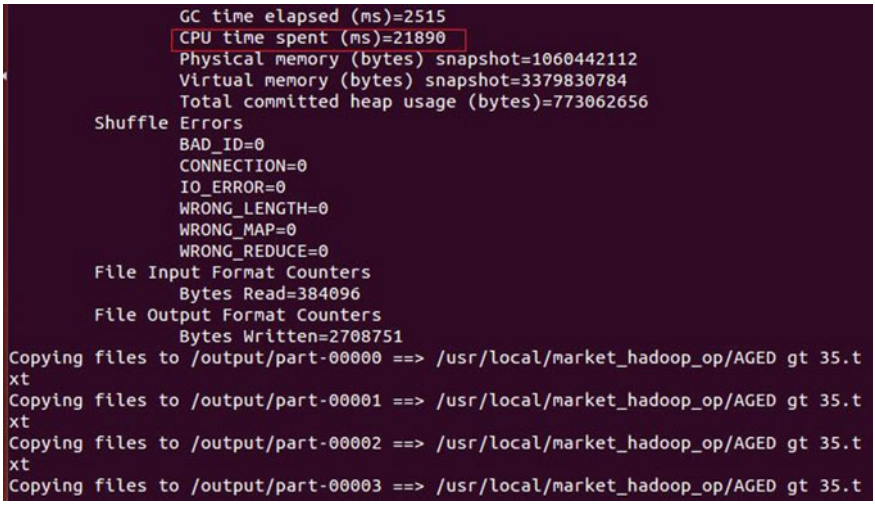

Fig. 4 Hadoop implementation using FIMMA

7 Conclusion

FIMMA technique is used to defeat the issues which are available in existing methods like parallel mining and load adjusting algorithms. In this approach, FIMMA algorithm (Hash-based) is proposed using MapReduce. The Comparison between existing method and proposed methods shows that there is up to 60% reduction in time. Hash-based Apriori is most efficient for generating the frequent itemset than existing methods. Data partitioning and data distribution is done by using this proposed system. FIMMA stores the previous results in hash tables and therefore it reduces time. At each result stage, the new input will be compared with the hash table and if matches with hash table's stored result then vector bit is set to 1 and named as a frequent item. All the set vector's list will be combined by the reducer, modify it and gives the final result. The proposed system works efficiently on MapReduce stages and gives the final result, rather than using the FIUT algorithm and increases the performance.

References

1. Bechini, Alessio, Francesco Marcelloni, and Armando Segatori. "A MapReduce solution for associative classification of big data", Information Sciences, 2016.
2. X Zhou, Y Huang - Fuzzy Systems and Knowledge Discovery. An Improved Parallel Association Rules Algorithm Based on MapReduce Framework for Big Data", pp. 284–288, 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery.

3. Yaling Xun, Jifu Zhang, and Xiao Qin, FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce" IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, VOL. 46, NO. 3, pp. 313–325, MARCH 2016.
4. R. Agrawal, T. Imieli nski, and A. Swami, "Mining association rules between sets of items in large databases," ACM SIGMOD Rec., vol.22, no. 2, pp. 207–216, 1993.
5. S Deshpande, H Pawar, A Chandras, A Langhe, Data Partitioning in Frequent Itemset Mining on Hadoop Clusters"- 2016 – irjet.net, https://irjet.net/archives/V3/i11/IRJET-V3I11229.pdf.
6. M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on MapReduce," in Proc. 6th Int. Conf. Ubiquit. Inf. Manage. Common. (ICUIMC), Danang, Vietnam, 2012, pp. 76:1–76:8.
7. L. Zhou et al., "Balanced parallel FP-growth with MapReduce," in Proc. IEEE Youth Conf. Inf. Compute. Telecommun. (YC-ICT), Beijing, China, 2010, pp. 243–246.
8. Y.-J. Tsay, T.-J. Hsu, and J.-R. Yu, "FIUT: A new method for mining frequent itemsets," Inf. Sci., vol. 179, no. 11, pp. 1724–1737, 2009.
9. Kiran Chavan, Priyanka Kulkarni, Pooja Ghodekar, S. N. Patil, Frequent itemset mining for Big data", IEEE, Green Computing and Internet of Things (ICGCIoT), pp. 1365–1368, 2015.
10. M. Riondato, J. A. DeBrabant, R. Fonseca, and E. Upfal, "PARMA: A parallel randomized algorithm for approximate association rules mining in MapReduce," in Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.,Maui, HI, USA, pp. 85–94, 2012.
11. Wei Lu, Yanyan Shen, Su Chen, Beng Chin Ooi, "Efficient Processing of kNearest Neighbor Joins using MapReduce" 2012 VLDB Endowment 2150-8097/12/06, Vol. 5, No. 10, pp. 1016–1027.
12. [Online]. Available:www.vldb.org.
13. Shekhar Gupta, Christian Fritz, Johan de Kleer, and Cees Witteveen, "Diagnosing Heterogeneous Hadoop Clusters", 2012, 23 rd International Workshop on Principles of Diagnosis.
14. J. Dean and S. Ghemawat, "MapReduce: A flexible data processing tool," Commun. ACM, vol. 53, no. 1, pp. 72–77, Jan. 2010.
15. Yaling Xun, Jifu Zhang, Xiao Qin and Xujun Zhao, "FiDoop-Dp Data Partitioning in Frequent Itemset Mining on Hadoop clusters", VOL. 28, NO. 1, pp. 101–113, 2017.
16. Ramakrishnudu, T, and R B V Subramanyam. Mining Interesting Infrequent Itemsets from Very Large Data based on MapReduce Framework", International Journal of Intelligent Systems and Applications, Vol. 7, No. 7, pp. 44–49, 2015.
17. Jong So Park, Ming Syan Chen, Philip S, "An Effective Hash based Algorithm for mining Association rule", '95 Proceedings of the 1995 ACM SIGMOD international conference on Management of data, held on May 22–25, 1995, Pages 175–186.