Sādhanā (2020)45:169
https://doi.org/10.1007/s12046-020-01380-8

Distributed synthesized association mining for big transactional data

AMRIT PAL[1,2,*] and MANISH KUMAR[2]

[1]Department of Computer Engineering and Application, GLA University, Mathura, India
[2]Department of Information Technology, Indian Institute of Information Technology Allahabad, Prayagraj, India
e-mail: er.amritkabir@gmail.com

MS received 16 November 2017; accepted 15 May 2020

Abstract. Data is increasing rapidly day by day along with the transactional database. Dividing this data and storing it in a distributed manner is an effective way for storage and retrieval. Mining such distributed data with minimum dependence between sub-problems is a crucial task. Finding frequent itemsets and corresponding association rules is a big challenge while considering the aggregation in a distributed environment. To overcome these challenges, we propose a distributed frequent itemset generation and association rule mining algorithm using MapReduce programming model. The proposed scheme generates frequent itemset and mine association rules using a synthesized distributed technique. The rules are mined in a distributed manner, and then weights are assigned to subsets of data and association rules. A proper mixture of association rules that are generated in distributed manner is done using a weighted approach. This paper presents a novel MapReduce-based synthesis approach, which can work well over a distributed storage of large amount of data.

Keywords. Big Data; HDFS; MapReduce; Apriori; frequent itemset; association rule.

1. Introduction

The data that grows in that much amount that it becomes difficult to manage using conventional data management system is known as Big Data [1]. Facebook, Twitter, etc. generate data in petabytes [2] and the amount of data generated by the humans will reach about 44 zeta bytes by 2020 [3]; managing this huge amount of data is a challenge. The basic V's of the Big Data are volume, velocity, variety [4] and two more are variability and values [5]. These *V*s also impose challenges to the Big Data management and information retrieval. There are several solutions available for managing this huge amount of data based on a distributed environment. In this type of Big Data management, data is divided into chunks and stored in a distributed manner. Mining of data in a distributed manner requires data to be divided into parts and the mining algorithms independently run as sub-problems on this divided data; result of the mining is produced by combining the results of independently running sub-problems. If it is implemented efficiently, it can increase the performance of mining process and scalability, though there can be issues like partition of data, inter-sub-problem communication, management of local and global information. A comprehensive approach would be distribution of complete problem over a cluster, where tasks independently run on each node of the cluster and results from nodes are combined to generate the final result. Frequent itemset mining helps in finding the association in the transaction items. It has application in different areas like wireless sensor networks [6], cloud computing [7], market basket problems, etc.; there are several techniques available for finding association rules that follow a sequential approach [8–12]. Performing parallel mining consists of remodelling of the mining algorithms, so that it can work in a parallel or distributed environment [13]. The process involves dividing the mining algorithm into homogeneous independent sub-problems, so that sub-problems can be executed on different nodes independently. Remodelling the mining algorithm in a parallel environment may increase the performance and scalability of the system, but it raises issues like intermediate communication, central node management, workload distribution and mixture of results from the sub-problems. Failure of a sub-problem or a node can result in a decrease of the performance of overall system and its accuracy. MapReduce [14] is a distributed programming model that facilitates division of a problem into sub-problems and executes these sub-problems independently. MapReduce can perform distributed processing of data on a scalable cluster. It has features like high scalability, fault tolerance, etc.; adding and removing a processing node can be easily done in the MapReduce programming framework. MapReduce is a combination of two functions, namely Map and Reduce or mapper and

*For correspondence

Published online: 02 July 2020

reducer. The complete input data is partitioned into small blocks of data and stored at different nodes of the cluster. The map function first reads the data from the data block and generates a $<key, value>$ pair based on its functioning. The reduce function combines the results of all the mapper functions and generates the final result based on the result aggregation approach. MapReduce can provide efficient Big Data management and information retrieval, although remodelling of algorithm execution in the form of $<key, value>$ is a challenge [15]. Association estimation is an important task in data mining; algorithms like Apriori require a complete scan of databases and work iteratively to generate association rules. Converting these algorithms for running in parallel environment may increase the speed and scalability of the algorithm. It requires some basic modification in the algorithm that considers the distribution of tasks and data, over a distributed environment. As already discussed, MapReduce provides an efficient parallel environment, so that modifying the Apriori algorithm according to MapReduce may increase the speed and scalability of the overall system.

This work is focused on the formulation of distributed association rule mining. The proposed approach provides a scalable solution for association mining over large datasets. It considers the distribution of data over the cluster and assigns weights to different subsets of the data, and merges their results based on the weights, which assures scalability of the system. The iterative execution between map and reduce jobs is no longer required in the proposed scheme and it follows a sequential approach for MapReduce.

Section 2 describes the related work of frequent itemset mining and association rule generation. Section 3 describes a basic Apriori algorithm, MapReduce and Hadoop Distributed File System (HDFS). Section 4 introduces a frequent itemset and association rule mining approach, and its implementation in a distributed environment. Section 5 elaborates proposed MapReduce-based synthesis approach for the frequent itemset generation and association rule mining. Section 6 describes an example problem for association rule generation. Section 7 describes the experimental details and results and finally section 8 concludes the work.

2. Related work

Frequent itemset mining is the process of extracting frequent itemsets and further generation of the association rules. Apriori [11] and FP-Growth [12] are two well-known algorithms for finding frequent itemsets and generation of association rules. Distributed programming models like MapReduce provide scalable solutions for mining large datasets from a distributed storage. A distributed implementation of the available algorithms can provide the solution for frequent itemset mining in large datasets. This distributed remodelling of available algorithm will require extra overhead of division of task into subtasks, management of intermediate results and further aggregation of the results.

Proper independence between the subtasks is important for providing a scalable solution. To provide this, sharing of some intermediate information/data is required. Agrawal and Shafer [16] have proposed an approach that shares the local information iteratively as count sharing among different processors. Yang *et al* [17] proposed single reducer Apriori, an iterative implementation of the Apriori algorithm that uses MapReduce programming model. This approach uses count sharing among the mappers and a single reducer to combine the generated information. Each implemented iteration of the Apriori algorithm also requires a complete iteration of the MapReduce. Further, count-based approaches have been proposed by Lin *et al* [18], which are single-pass counting, fixed-pass counting and dynamic-pass counting, where MapReduce Apriori with DPS performs better than other passing techniques. Regarding the overall flow of these approaches, in each pass there is simple mapper counting and later the produced information is shared with the reducers to generate the global count for the items and share it with each mapper. This iterative communication between mapper and reducer continues until there are frequent itemsets in the dataset. The number of iterations between the mappers and reducers depends on the size of frequent itemset produced. The same procedure can also be considered for mining frequent itemsets in a cloud-based environment where data resides over a cloud storage [19]. Similarly, parallel algorithms based on iterative flow between mapper and reducer like MR-Apriori have been proposed [20–22]. Another type of parallel approaches that are tree based has also been proposed, which is basically a distributed implementation of the FP-Growth (Frequent Pattern-Growth) algorithm. Different algorithms have been proposed for mining frequent itemsets using FP-Tree in a distributed manner; Haoyuan Li proposed the Parallel FP-Growth (PFP) [23] algorithm, which uses the MapReduce programming model. It generates frequent itemsets using a parallel counting and maintains two lists, one for count maintenance and one for grouping of the items. Grouped items can be processed by different processors, for which group list needs to be shared among the processors. This algorithm requires several iterations of the MapReduce over the dataset and also puts an extra overhead of grouping the items.

A Parallel Randomized Algorithm for Approximate Association Rule Mining (PARMA) [25] approach exists for mining of the approximated association rules; an open source implementation of the approach also exists as Apache SAMOA [26]. PARMA requires random sampling of the data; the size of the sample varies based on the accuracy requirement of the user. Specific parameters need to be tuned to make the model adaptive for the parallel computing model. The proposed approach does not require such parameter tuning and adapts well with the distributed programming model.

Xun *et al* [24] proposed the FiDoop algorithm, which uses the FIUT algorithm and MapReduce. FiDooP requires three iterations of MapReduce, in which the first two iterations are for the creation of a set of maximal size frequent itemsets. In the third iteration of MapReduce, mappers decompose the maximal frequent itemsets generated in previous iterations and build FIU-Tree. Later, trees having the same number of items are assigned to a single reducer for further decomposition or information extraction. The algorithm requires itemsets ordering, which is not always present in the real time data [6, 27]. Load balancing is required in the third iteration, as it includes the decomposition process, which is recursive in nature and can result in uneven distribution of load on nodes executing mappers. Although the algorithm shows better performance than the PFP algorithm, which requires multiple iterations of the MapReduce, PFP was proposed for query recommendation system and does not follow any specific order in the input query as it is hard to maintain such order in user input queries.

There are several concerning aspects for mining association rules in parallel, like iteration management, inter-mapper communication and intermediate output handling, as shown in table 1. This paper has made a novel attempt to handle these challenges for increasing the scalability of the frequent itemset and the association rule mining process using a MapReduce-based synthesis approach while ensuring no iterative approach between map and reduce. Existing works have considered the Apriori algorithm and applied it using MapReduce, which is a very iterative model. On the contrary, the proposed approach reduces the iterations between the mapper and the reducer to one.

3. Preliminary

3.1 *Apriori algorithm*

In a transactional database, Apriori algorithm [11] can be used to generate the frequent itemsets. The basic approach of Apriori is to scan each transaction and generate the frequent itemsets based on occurrence of an item (or set of items) in the dataset. There are two types of itemsets generated during the algorithm execution, large q-itemset and candidate q-itemset itemsets, represented by L and C, respectively, where q is the length of the itemset. The algorithm starts with generating C_1, which is a combination of itemsets of size 1 and there corresponding support counts. Large 1-itemset(L_1) is generated using candidate itemset C_1 by removing the itemset having support count less than the minimum support count. Joined itemsets generated by combining the itemset of previous large set L_1 are the itemsets used for C_2. After generation of C_2, L_2 is generated by eliminating infrequent itemset. Subsequently, in the q^{th} iteration, the algorithm performs in two phases, calculating L_{q-1} and generating candidate set C_q. The algorithm terminates when there are no more frequent itemsets to generate.

3.2 *MapReduce and HDFS*

The problem of mining for Big Data, i.e. dividing the problem into sub-problems and processing data independently, can be overcome using the MapReduce programming model. It provides facilities for processing data independently with a large scalability. There are two primary tasks for processing the Big Data: the first is data storage and the second is retrieval of useful information. HDFS [28, 29] is a data storage system that can store data in a distributed manner. Data is stored in blocks form by default block of size 64 MB each. There can be a number of nodes in the cluster to process the data. This distributed environment consists of namenodes, datanodes, secondary namenode, task trackers and job trackers. Information of the block storage is maintained by the namenode, and block mapping can be done using this information. There can be a number of datanodes that can exist in a cluster based on the scalability requirement of a cluster, and storage of the blocks is done on the datanodes. The secondary namenode works as a check point node for maintaining the validity of the namenode. The jobs are submitted to the job tracker and it splits the job and assigns each job to a task tracker. Task trackers execute jobs submitted on a datanode by fetching the data from that datanode. A task tracker has a fixed number of slots for execution of a job. The job tracker monitors job execution by each task tracker. The MapReduce programming model facilitates the execution of the tasks in a distributed manner. It comprises two main functions: map and reduce functions. The mapper takes data blocks as input and produces $<key, value>$ pair as output; the reducer receives this pair as input and processes it for further results.

4. Synthesized approach

Transactional database is a collection of transactions that is a combination of a transaction reference and a transaction (set of items). Consider a set of items $I = I_1, I_2, I_3, \ldots, I_n$ and D as a transactional database that contains transactions $T_1, T_2, T_3, \ldots, T_N$, where $T_i \subseteq I$, n is the total number of items and N is the total number of transactions. Frequent itemset mining is the process of identifying itemsets that have a higher probability of occurrence; further, it can lead to finding the association between two itemsets, which in turn results in better decision making. Basic entities used in frequent itemset mining are itemset support and confidence, where support is the probability of occurrence of an itemset

Table 1. Overview of some existing frequent itemset mining approaches.

Sl. no.	Approach	Issues
1	Count-exchange-based approach [18]	Iterative and data exchange required
2	Single-reducer-based approach [17]	Iterative candidate set exchange
3	A cloud-based implementation of MapReduce-based Apriori [19]	Multiple scans over the cloud, data exchange over cloud
4	Candidate set partition over the cluster [20]	Iterative and data exchange
5	Parallel Apriori [21]	Iterative and data exchange
6	Parallel FP-Growth [23]	Information exchange and iterative over MapReduce
7	FiDoop [24]	First *k* iterations for forming *k*-size itemset and in-memory FP-Tree handling
8	Distributed Apriori over a cloud [22]	Iterative and data exchange

in the transactional database and confidence is the conditional probability that occurrence of an itemset can result in occurrence of another itemset. Calculation of these two entities results in formulation of the association rules for that particular transactional database.

The frequent itemset mining is commonly used for market basket analysis, where each basket is a list of items that a customer has purchased together. The problem can be formulated as the estimation of joint probability values of variable $Y = Y_1, Y_2, Y_3, \ldots, Y_n$ that occur more frequently in a database [30, 31].

Y_j is an item for any i^{th}observation or transaction. $Y_j \in \{0, 1\}$ and $y_{ij} \in \{0, 1\}$ represent the presence of an item in i^{th} transaction. Variables that have a joint value of 1 together are considered to be purchased together. The complete goal of frequent itemset and association mining is to find the prototype Y-values $v_1, v_2, \ldots, v_L$ for which $P(v_l)$ (where v_l is the set of frequent items) is relatively large. A simple estimator of $P(v_l)$ is fraction of transactions where $y = v_l$. Consider the set of all possible support values that a j^{th} variable can attain as S_j; then, for $s_j \subseteq S_j$, the goal is to find conjunctive rule probability for each subset $s_1, s_2, \ldots, s_\phi$ where it is relatively large:

$$P[\cap_{j=1}^{\phi}(Y_j \in s_j)]. \tag{1}$$

A dummy variable technique can be used and a new set of dummy variables $Z_1, Z_2, \ldots, Z_K$ is created for attainable values of each Y_j variable. Hence, the total number of dummy variables will be $K = \sum_1^{\phi} |S_j|$, where $|S_j|$ is the total number of distinct values attainable by Y_j. Z_k can be assigned 0 or 1 based on the assignment of Z_k to its associated variable Y_j. The estimated values for support in database can be calculated using Eq. (2):

$$P\left[\prod_{k \in K}(Z_k = 1)\right] = \frac{1}{N}\sum_{i=1}^{N}\prod_{k \in K} z_{ik} = Supp(K) \tag{2}$$

where z_{ik} is the value that variable Z_K will have in i^{th} case and $Supp(K)$ is the support value for an itemset K. An association rule is represented as $A \rightarrow B$ where A is antecedent and B is consequent. The confidence for the rule can be calculated as

$$C(A \rightarrow B) = \frac{Supp(A \rightarrow B)}{Supp(A)}. \tag{3}$$

Big Data requires distribution of complete data over a distributed storage. Calculation of support and confidence requires complete information about the dataset, so maintaining the independence between sub-problems is a big challenge. Some parameters that need to be formulated are distributed support and distributed confidence. The total number of transactions will be divided over distributed storage and then the result of calculating support using Eq. (2) will be mapped to Eq. (4) with reference to its distribution, where N_m is the number of transactions in the m^{th} subset of the dataset and M is the total number of data partitions. Each of these support value can be used for generation of association rules from locally available data. More generalization of the support calculation in distributed manner is described in the next section using MapReduce. Synthesizing these rules requires a weighted approach for generation of final confidence using Eq. (5), which can converge to Eq. (6):

$$Supp_m(K) = \left\{\frac{1}{N_m}\sum_{i=1}^{N_m}\prod_{k \in K} z_{ik}\right\}, \tag{4}$$

$$C(A \rightarrow B) = w_1C_1 + w_2C_2 + \ldots.. w_MC_M, \tag{5}$$

$$C(A \rightarrow B) = \sum_{i=1}^{M} w_iC_i. \tag{6}$$

There are several issues while applying this technique, such as the following.

4.1 *Management of the global and local information*

There is a requirement of properly sharing the local variables/parameters for generating the global information.

4.2 *Proper mixture of the distributed information*

Considering nodes as the distributed entities, each node will process the local data and produce local results. There is a need of an efficient mixture technique, so that these results can be combined.

4.3 *Proper iteration of the mining process*

A number of iterations are required for finding the frequent itemsets from transactional databases. In case of distributed data, there can be two approaches while considering the iterations. The first is scanning dataset, then generating global information, sharing this information and again scanning for generating next frequent itemsets. This process is iterated till frequent itemsets are generated. The second is an iterative scan of local dataset and then combining the information to get the resultant frequent itemsets.

4.4 *Formulation of the weighted approach*

Since the amount of data at each node can vary in size, weightages of the results from a particular node are not equal. To overcome this problem, there is a requirement of the weighted approach. It will provide proper weight to results generated by different nodes.

5. MapReduce synthesized approach

Conversion of sequential mining algorithm according to the MapReduce programming framework requires some basic modification in the calculation of frequent itemsets based on the minimum support count. Figure 1 shows a general idea about implementation of synthesized approach using MapReduce programming model. The inputs are the transaction containing the items in a single transaction. The complete dataset is divided and works as an input to different mappers. Mappers process this dataset subset, produce frequent itemsets locally and then generate association rules from them. Each mapper passes association rule, confidence and a dataset subset reference. This output of the mappers is then used as the input to the reducer, which further processes it to generate final association rules with their combined confidence. The common notations used are listed in table 2.

5.1 *Map function*

The transactional database is stored over a distributed storage using HDFS, which provides facility to store data over a distributed environment in the form of blocks. Map function starts with mapping the data or transactions distributed over this distributed storage, and processing each transaction line by line. The mapper reads a small part of data, for example a block; then it processes that data. The processing by the mapper consists of applying the Apriori algorithm on that part of data as shown in Algorithm 1. Each mapper first applies Apriori on its own part of data; later it generates frequent itemsets locally and association rules with confidence as $<Rule, Confidence, Dataset\_ID>$ based on the minimum support value. Proper sharing of information can be assured as each mapper writes to a distributed storage, which is directly accessed by the reducer.

Algorithm 1 Mapper

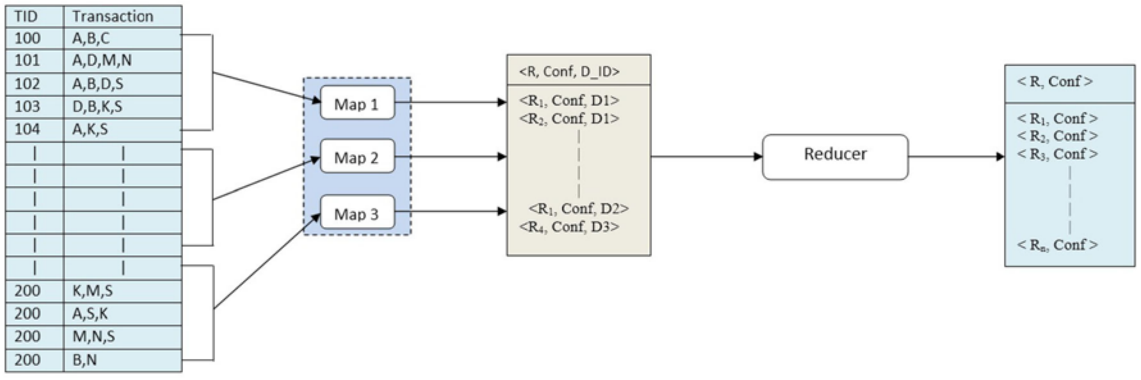
```
Input: key, itemset in transaction itemset $t_i$
Output: datasetID, frequentitemset, confidence
    Initialisation :
 1: $D_i$ dataset partition
 2: $C_k$ is the candidate itemset of size k
 3: $L_k$ frequent itemset of size k
 4: $L_1$= frequent itemset of size 1
 5: while $L_k != \emptyset$ do
 6:     $C_{k+1}$= candidate generation from $L_k$
 7:     for each transaction $t_j \in D_i$ do
 8:         Count for itemset increase by one if t contain the itemset
 9:         $L_{k+1}$= candidate itemsets in $C_{k+1}$ with min-support
10:     end for
11: end while
12: for each frequent itemset in $L_{k+1}$ do
13:     Write datasetID, Rule,confidence
14: end for
```

5.2 *Reduce function*

As mappers produce output as a triplet $<Rule, Confidence, Dataset\_ID>$, reduce function or reducer processes these intermediate outputs. The reducer receives a set of rules R and combines the rules generated by each mapper. For synthesizing confidence values, the first rule, gravity for each rule R_i, represented by $G(R_i)$, is calculated using Eq. (7):

$$G(R_i) = \frac{Count(R_i)}{|R|}. \tag{7}$$

For each rule generated by the mappers, rule gravity is calculated by the reducer function. Weights for each rule are calculated and treated as an intermediate output. These weights are used to calculate the weights for each subset of the dataset. The next step is to calculate the weights for each dataset subset processed by the mapper. Dataset



TID	Transaction
100	A,B,C
101	A,D,M,N
102	A,B,D,S
103	D,B,K,S
104	A,K,S
|	|
|	|
|	|
|	|
|	|
|	|
200	K,M,S
200	A,S,K
200	M,N,S
200	B,N

Figure 1. MapReduce-based scenario for extraction of the frequent itemsets and generation of the association rules.

Table 2. Notations for MapReduce synthesized approach.

Notation	Description
D	Transactional database
D_i	ith subset of the dataset D
m	Number of mappers
N	Total number of subsets of the dataset
R	Set of rules
R_i	ith rule
$G(R_i)$	Gravity of rule i
W_{D_i}	Weight for subset D_i
$Count(R_i)$	Count for rule R_i over the entire dataset
L_k	k-length frequent itemset
C_k	k-length candidate itemset
$NormD_i$	Normalized weight for subset D_i
$Conf(R_i)$	Confidence for rule R_i
$Supp(R_i)$	Support for rule R_i
M_{D_i}	Set of rules from ith subset of the dataset

weight decides the importance of a dataset subset for calculation of the confidence for a particular frequent itemset. The importance of a dataset is a weighted combination of the frequent itemset that exists in this dataset. From dataset D_i, mappers generate the set of rules M_{D_i}.

$$W(D_i) = \sum_{\forall R_j \in M_{D_i}} Count(R_j)G(R_j). \tag{8}$$

The next step is calculating the confidence for each frequent itemset by linearly combining intermediate outputs. Confidence for each frequent itemset is calculated based on the transactions available in the subset of the dataset. Weights for each dataset are calculated and need to be normalized; for normalization of the weights, a simple method is adopted. Weights of each dataset is divided using

$$Norm_{D_i} = \frac{W_{D_i}}{\sum_{\forall D_i \in D} W_{D_i}}. \tag{9}$$

These normalized weights are used for linearly combining the confidence calculated by the different mappers. Normalized weights provide a weighted summation of the different confidence for each rule calculated by the mappers. The combined synthesized confidence of a rule can be obtained using Eq. (10), which ensures the proper mixture of results from different mappers:

$$Conf(R_i) = \sum_{\forall D_j \in D} Norm_{D_j} \times Conf_{D_j}(R_i). \tag{10}$$

Algorithm 2 Reducer

```
Input: $R, D, Conf_{D_j}(R_i)$
Output: $R, Conf(R_i)$
 1: R set of rules generated by the mappers
 2: D set of data partitions
 3: for each rule $R_i \in R$ do
 4:     calculate $G_{R_i}$ using Equation 7
 5: end for
 6: for each rule $D_i \in D$ do
 7:     calculate $W_{D_i}$ using Equation 8
 8: end for
 9: for each rule $D_i \in D$ do
10:     calculate $Norm_{D_i}$using Equation 9
11: end for
12: for each rule $R_i \in R$ do
13:     calculate $Conf(R_i)$using Equation 10
14: end for
15: return $R_i, Conf(R_i)$
```

6. Example problem

Applying the Apriori algorithm to a dataset requires the complete dataset to be considered at once; this condition cannot be fulfilled while applying this algorithm on a distributed dataset. Apriori algorithm calculates the support and confidence for different set items based on the value of minimum support. An association between two disjoint itemsets can be denoted as $A \rightarrow B$. Based on the Apriori algorithm, the support S and confidence C can be calculated.

The transactional database in table 3 has been considered for applying the synthesized Apriori algorithm. First, the complete dataset is divided into two subsets D_1 and D_2. These two subsets of the dataset can be of different sizes. The Apriori algorithm is applied independently on both the subsets for generation of the rules. The synthesized process captures the conditional dependence in the form of confidence from each subset of the data.

A set R of corresponding association rules from frequent itemset, generated from D_1 and D_2, is $R = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11}, R_{12}, R_{13}\}$. Frequent itemsets from different subsets of the dataset D are as follows: from $D_1 = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_9, R_{10}\}$, from $D_2 = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11}, R_{12}, R_{13}\}$.

The first step is to calculate the gravity of the rules generated from frequent itemsets using Eq. (7). Table 4 shows the rules and their confidence calculated for each subset of the data. Table 5 shows the rules generated from different datasets and their respective counts. Each rule will be assigned a gravity value based on its counts. This value is then used for calculating the weight for the dataset

Table 3. Transaction datasets D, D_1 and D_2.

Transaction ID	Transaction
D	
T01	$I_1I_2I_5$
T02	I_2I_4
T03	I_2I_4
T04	$I_1I_2I_4$
T05	I_1I_3
T06	I_2I_3
T07	I_1I_3
T08	$I_1I_2I_3I_5$
T09	$I_1I_2I_3$
D_1	
T01	$I_1I_2I_5$
T02	I_2I_4
T03	I_2I_4
T04	$I_1I_2I_4$
T05	I_1I_3
D_2	
T06	I_2I_3
T07	I_1I_3
T08	$I_1I_2I_3I_5$
T09	$I_1I_2I_3$

Table 4. Generated rules.

Frequent itemset	Confidence
From dataset D_1	
$1 \to 2$	0.666667
$1 \to 2$	4 0.5
$12 \to 5$	0.5
$1 \to 3$	0.333333
$1 \to 4$	0.333333
$1 \to 5$	0.333333
$2 \to 3$	0.25
$2 \to 4$	0.5
$2 \to; 5$	0.25
From dataset D_2	
$1 \to 2$	0.666667
$12 \to 3$	1
$12 \to 5$	0.5
$123 \to 5$	0.5
$1 \to 3$	1
$13 \to 5$	0.333333
$1 \to 5$	0.333333
$2 \to 3$	1
$23 \to 5$	0.333333
$2 \to 5$	0.333333
$3 \to 5$	0.25

subset. Each subset of the dataset will get a weight associated with it using Eq. (8) as $W_{D_1} = 2.07692$, $W_{D_2} = 2.23076$. Normalization of weights of the two datasets subset can be calculated using Eq. (9) as $Norm_{D_1} = 0.4821428$, $Norm_{D_2} = 0.5178571$. These normalized weights can be used for calculating the confidence for each rule using Eq. (10) as $Conf(R_1) = 0.6666667$, $Conf(R_2) = 0.678571432$ and $Conf(R_3) = 0.160714289$. In this way, confidence for each rule can be calculated; rules that satisfy the minimum criteria for selection will be selected for decision making.

7. Experimental results and analysis

For implementing the MapReduce-based synthesis approach, a Hadoop cluster has been configured using four datanodes and one namenode. The architecture of the cluster has been shown in figure 2. The cluster set-up has been done using five systems with Intel(R) Core(TM)i3-3220 processors, Ubuntu 14.04 as operating system, 6-GB RAM and star-Ethernet-based local area connectivity based on SSH Server. Each node in the cluster is configured using Hadoop 2.6.4 with 10 mappers and replication factor as 3. The cluster requires a benchmarking test, which is done using the Hadoop benchmarking techniques. The complete analysis of the proposed synthesized approach is done using benchmark datasets [32] as described in table 6. The

Table 5. Rule set R.

R	Rule	Rule count
$R1$	$1 \to 2$	2
$R2$	$1 \to 3$	2
$R3$	$1 \to 4$	1
$R4$	$1 \to 5$	2
$R5$	$2 \to 3$	2
$R6$	$2 \to 4$	1
$R7$	$2 \to 5$	2
$R8$	$3 \to 5$	1
$R9$	$12 \to 4$	1
$R10$	$12 \to 5$	2
$R11$	$12 \to 3$	1
$R12$	$13 \to 5$	1
$R13$	$23 \to 5$	1

experimental set-up, analysis and the results discussion include the following:

- set-up of a Hadoop cluster for data processing,
- analysis of the proposed distributed approach for errors as compared to the sequential approach,
- analysis based on the previously used dataset in the literature,
- a scalability analysis of the synthesized approach,
- analysis of the mapper and reducer phases of the synthesized approach,

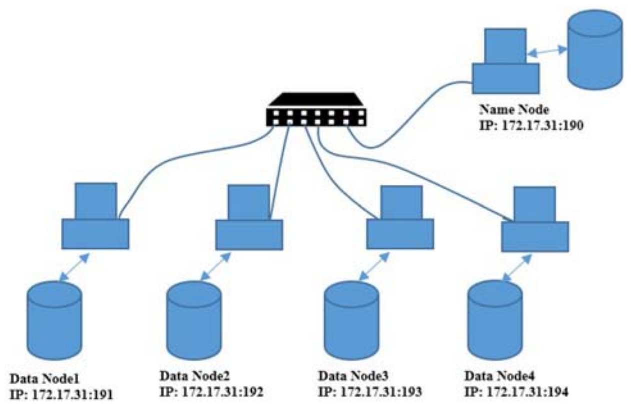


Figure 2. System architecture of Hadoop cluster with four datanodes and one namenode.

- a comparative analysis of the proposed approach with the existing approaches,
- evaluation of the synthesized approach with different dataset sizes is done and compared to the other approaches.

The process of frequent itemset mining starts with submission of the job at namenode, which then submits it to mappers running independently. The experiment has been done by considering the transactional dataset T10I4D100K [32], which contains a sequence of itemsets in each transaction. The proposed approach works well on the divided dataset with very small proportion of error. Different error measures are used to compare the results of MapReduce-based approach and sequential approach. Absolute error is calculated using Eq. (11):

$$Error_{abs}[i] = |Conf_{complete}[i] - Conf_{distributed}[i]| \tag{11}$$

where $Conf_{complete}[i]$ is the confidence for the i^{th} rule while considering the complete dataset, and the confidence for the i^{th} rule is $Conf_{distributed}[i]$ while considering the distributed dataset. The average error is calculated over all rules of a specific size using Eq. (12):

$$Avg_{error_j} = \frac{1}{N_j} \sum_{\forall i | size(i)=j} Error_{abs}[i] \tag{12}$$

where N_j is the number of rules of size j and $size(i)$ is the number of items in the i^{th} rule. Further, to get the spread out of the errors for rules, standard deviation is calculated using Eq. (13):

$$\sigma_j = \sqrt{\frac{1}{N_j} \sum_{\forall i | size(i)=j} (Avg_{error_j} - Error_{abs}[i])^2}. \tag{13}$$

Comparison of errors for different sizes of frequent itemset has been done, and table 7 illustrates the errors generated by the proposed synthesized approach. Further, for testing the MapReduce-based approach, analysis has been done on different datasets, as shown in table 8. In this table the average error and standard deviation has been calculated again using the absolute error while considering the total number of rules irrespective of their size. Results show marginal errors while considering these different datasets [32] over a distributed environment.

Next, we have used the synthetic data generator for generating the transaction itemsets using [33]. The generated transaction is then stored on HDFS.

For testing the MapReduce-based synthesis approach, an experiment has been done, which uses one million transactions with a varying value of support count as absolute support value. The purpose of the analysis is to show that the time required and the association rules generation pattern follow the same pattern as it would be in a sequential approach, so that there would be no ambiguity about the behaviour and results of the proposed algorithm. The graphs in figure 3 demonstrate that increase in absolute support count results in reduction of time required to find association rules among the itemsets. The time measured is the average time taken by each mapper to perform the mining task on the subset of the complete dataset submitted to that particular mapper. The number of frequent itemsets and their corresponding rules, generated by the mapper, decrease with increase in the absolute support count. These two results follow the same nature as those in the sequential approaches. Our analysis shows that a scalable and less

Table 6. Datasets used for the analysis.

Dataset	Transactions	Items
Accident	8416	119
Chess	3196	75
KDD99	1000000	135
Mushrooms	8416	119
PAMAP	1000000	141
PowerC	1040000	140
Pumsb	49046	2113
Susy	5000000	190
US Census	1000000	396
T10I4D100K	100000	870

Table 7. Frequent errors based on itemset size.

FI size	Average error	Standard deviation
2	0.014143863	0.010997393
3	0.012870628	0.009925736
4	0.011304772	0.002273694
5	0.008863346	0.007304795

Table 8. Frequent errors based on itemset size.

Dataset	Average error	Standard deviation
Accident	0.06498348	0.01368334
Chess	0.07463866	0.07802999
KDD99	0.08515631	0.0648506
Mushrooms	6.15E-003	0.008191344
PAMAP	0.19581582	0.12320907
POWERC	0.0382273580	0.014850
PUMSB	0.08652966	0.09622266
SUSY	0.01357976	0.01196921
US Census	0.162436541	0.087418035

iterative approach can be achieved for mining of the frequent itemsets and generation of the association rules with minimal errors.

Scalability of the system is tested by increasing the number of nodes in the cluster. A synthetic dataset has been used for this process, and 10 million transactions have been used for testing the scalability of the system. In the ideal case, processing time should decrease in the same ratio in which the resources are added to the cluster. However, this is a theoretical assumption. In real time, the system does not behave in this way. Figure 4 shows the scalability results of the proposed algorithm with increasing number of nodes. Scalability factor is used for this analysis, which is given in Eq. (14):

$$SF = \frac{F_T}{C_T} \tag{14}$$

where SF is scalability factor, F_T is the time required for the mining process in first deployment of the system (which is a 1 node in the considered case) and C_T is the time required in the current deployment (value will change as the nodes increases).

As the proposed algorithm reduces the map reduce iteration to one, an analysis is required for the map and reduce phase execution time. Figure 5 shows the percentage of time required in map and reduce phases of the algorithm with varying minimum support value for the dataset. We have done a comparative analysis of the synthesized approach with the available approaches as shown in figure 6 using the T40I10D100K [32] dataset with a varying minimum support value.

It has been observed that due to reduction in the communication and MapReduce iteration, the proposed approach performs well as compared with other approaches. The tree-based approaches PFP and FiDoop perform better than the proposed approach when the minimum

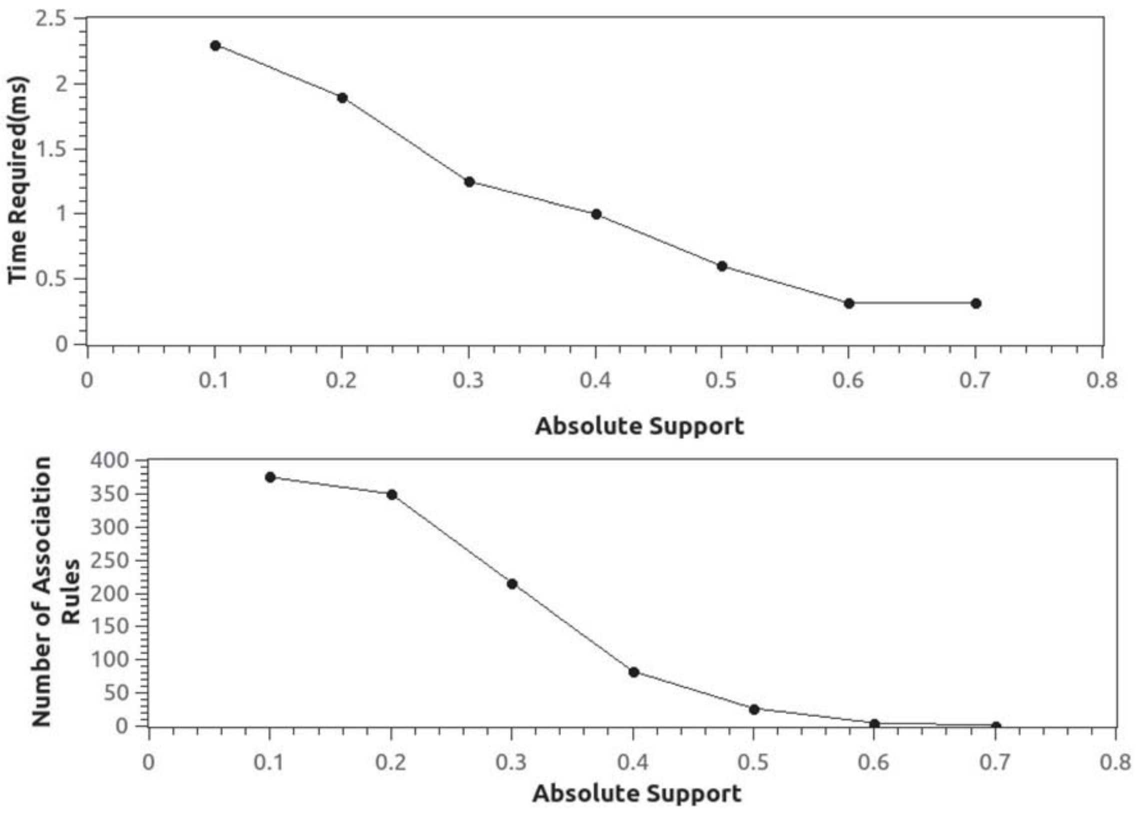


Figure 3. Average time required by each mapper and corresponding absolute support and number of rules generated with respect to the absolute support.

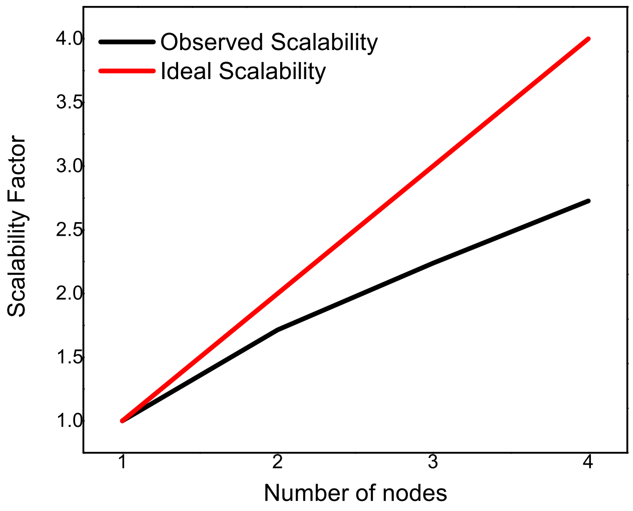


Figure 4. Scale-up of the system ideal vs. actual.

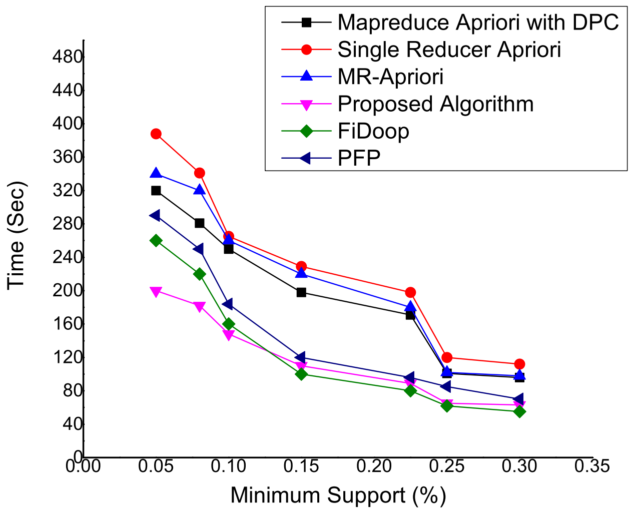


Figure 6. Comparative analysis of proposed approach.

support value is relatively high. The synthesized approach performs well for small value of the minimum support. The reason is that the generated tree in case of the tree-based approaches requires more time to process and to maintain the tree structure, while the proposed approach follows the strict MapReduce parallel scenario of managing everything on the disk. The performance of the proposed algorithm does not get affected by the increased size, because it does not have to maintain any data structure like *Frequent Pattern-Tree* in memory.

To support this argument well, we have further done time-based analysis with different sizes of the datasets. The FiDoop shows better performance with a comparatively lesser number of transactions as shown in figure 7. This is because as the size of the dataset increase the size of the generated FP-Tree also increases, which puts an extra overhead for processing and it has to iterate *k* times over mappers to generate the *k*-frequent itemset. In contrast the proposed algorithm does not require any such data structure to be maintained in the memory, so it provides a better performance with large datasets. The overall results show that the proposed algorithm performs well as compared with existing approaches, requires no inter-mapper communication and provides significance scalability for large dataset applications. Not only in data processing, due its compression capability, it can be used in application where communication cost is a very crucial resource like in wireless sensor networks.

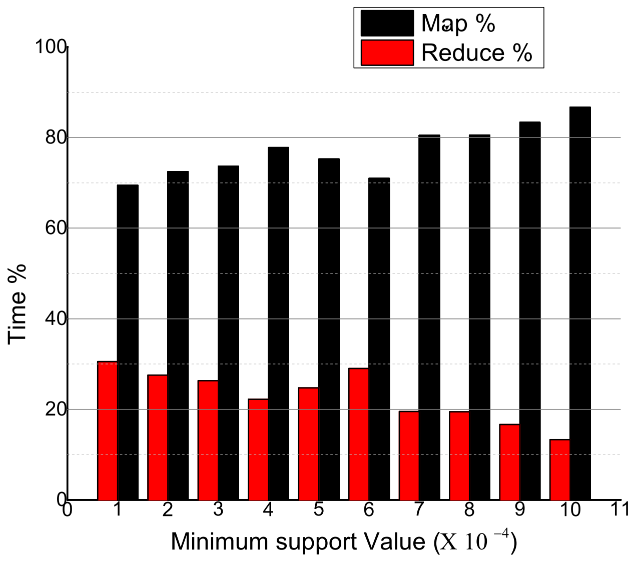


Figure 5. Map and reduce phase percentage ratio.

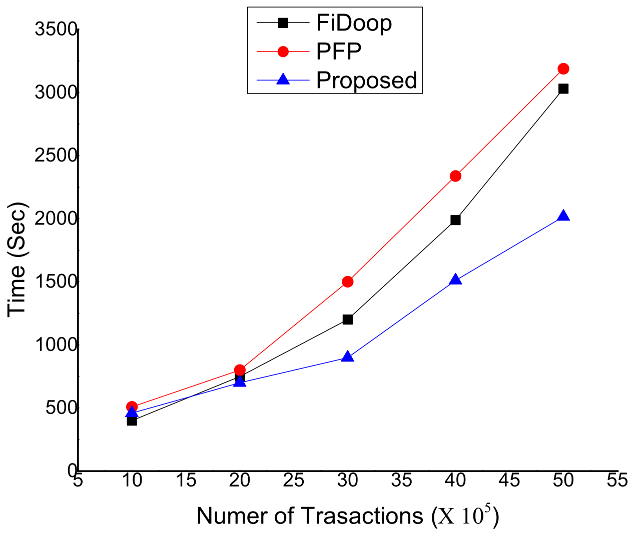


Figure 7. Scalability comparison of proposed approach.

8. Conclusion

Generation of frequent itemsets and subsequently establishing an association among them can achieve a better decision making. An effective synthesis scheme has been proposed, which yields required mixture capabilities for estimation of values for the desired variables where HDFS works as a base for information maintenance and MapReduce provides necessary scalability to the system. The overall approach has been tested on some previously available datasets that are vastly used in similar approaches. Scalability testing has been performed on synthetic data generated by the synthetic data generator. The proposed approach performs well as compared with the previously available approaches. The complete approach is limited to transaction data; further, we would like to apply pre-processing techniques through mappers, which will produce transactional data from raw data, and then apply association mining. In the immediate future, we would like to implement the approach for online association rule generation in a real time environment for a data stream. We would also like to implement the synthesis approach for sensors pattern mining in an internet of things eco-system, where this technique will be helpful due to its compression and small communication capabilities.

References

[1] Wu X, Zhu X, Wu G Q and Ding W 2014 Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering* 26(1): 97–107

[2] DeMers, J 2015 Why Facebook is making big data available to select partners. *Forbes*, retrieved from http://www.forbes.com/sites/jaysondemers/2015/03/25/why-facebook-is-making-big-data-available-to-select-partners/#24f4d0422966

[3] Turner V 2014 *The digital universe of opportunities: rich data and the increasing value of the Internet of things*. Retrieved October 26, 2016, from http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[4] Laney D 2001 *3D Data management: controlling data volume, velocity and variety*. META Group Research Note 6, 70

[5] Fan W and Bifet A 2013 Mining Big Data: current status, and forecast to the future. *ACM SIGKDD Explorations Newsletter* 14(2): 1–5

[6] Rashid M M, Gondal I and Kamruzzaman J 2017 Dependable large scale behavioral patterns mining from sensor data using Hadoop platform. *Information Sciences* 379: 128–145

[7] Anitha R, Mukherjee S 2015 MaaS: fast retrieval of data in cloud using metadata as a service. *Arabian Journal for Science and Engineering* 40(8): 2323–2343

[8] Hipp J, Güntzer U and Nakhaeizadeh G 2000 Algorithms for association rule mining: a general survey and comparison. *ACM SIGKDD Explorations Newsletter* 2(1): 58-64.

[9] Seol W S, Jeong H W, Lee B and Youn H Y 2013 Reduction of association rules for Big Data sets in socially-aware computing. In: *Proceedings of the 16th IEEE International Conference on Computational Science and Engineering (CSE)*, pp. 949–956

[10] Han J 2005 *Data mining: concepts and techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[11] Agrawal R and Srikant R 1994 Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1215*, pp. 487–499

[12] Han J, Pei J, Yin Y and Mao R 2004 Mining frequent patterns without candidate generation: a frequent-pattern tree approach *Data Mining and Knowledge Discovery* 8(1): 53–87

[13] Ordonez C, Mohanam N, Garcia-Alvarado C 2014 PCA for large data sets with parallel data summarization. *Distributed and Parallel Databases* 32(3): 377–403

[14] Dean J, Ghemawat S 2008 MapReduce: simplified data processing on large clusters. *Communications of the ACM* 51(1): 107–113

[15] Agrawal D, Das S, El Abbadi A 2011 Big data and cloud computing: current state and future opportunities. In: *Proceedings of the 14th International Conference on Extending Database Technology*, 530–533

[16] Agrawal R, Shafer J C 1996 Parallel mining of association rules: Design, implementation, and experience *IBM Thomas J. Watson Research Division*

[17] Yang X Y, Liu Z, Fu Y 2010 MapReduce as a programming model for association rules algorithm on Hadoop. In: *Proceedings of the 3rd International Conference on Information Sciences and Interaction Sciences (ICIS)*, pp. 99–102

[18] Lin M Y, Lee P Y, Hsueh S C 2012 Apriori-based frequent itemset mining algorithms on MapReduce. In: *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, p. 76

[19.] Chang X Z MapReduce-Apriori algorithm under cloud computing environment. In: *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 2, pp. 637–641

[20] Lin X 2014 MR-apriori: association rules algorithm based on MapReduce. In: *Proceedings of the 5th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 141–144

[21] Li N, Zeng L, He Q, Shi Z 2012 Parallel implementation of apriori algorithm based on MapReduce. In: *Proceedings of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD)*, pp. 236–241

[22.] Guo J, Ren Y G 2013 Research on improved A Priori algorithm based on coding and MapReduce. In: *Proceedings of the 10th Conference on Web Information System and Application (WISA)*, pp. 294–299

[23] Li H, Wang Y, Zhang D, Zhang M and Chang E Y 2008 Pfp: parallel fp-growth for query recommendation. In: *Proceedings of the 2008 ACM Conference on Recommender Systems*, pp. 107–114

[24] Xun Y, Zhang J and Qin X 2016 Fidoop: parallel mining of frequent itemsets using MapReduce. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46(3): 313–325

[25] Riondato M, DeBrabant J A, Fonseca R and Upfal E 2012 PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, October, pp. 85–94

[26] Morales G D F and Bifet A 2015 SAMOA: scalable advanced massive online analysis. *Journal of Machine Learning Research* 16(1): 149–153

[27] Holt J D and Chung S M 2007 Parallel mining of association rules from text databases. *The Journal of Supercomputing* 39(3): 273–299

[28] Shvachko K, Kuang H, Radia S and Chansler R 2010 The Hadoop distributed file system. In: *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10

[29] Javed A and Khokhar A 2004 Frequent pattern mining on message passing multiprocessor systems. *Distributed and Parallel Databases* 16(3): 321–334

[30] Wu X, Zhang S 2003 Synthesizing high-frequency rules from different data sources. *IEEE Transactions on Knowledge and Data Engineering* 15(2): 353–367

[31] Friedman J, Hastie T, Tibshirani R 2001 The elements of statistical learning. In: *Springer Series in Statistics*, vol. 1. Berlin: Springer

[32] Fournier-Viger P 2008 *SPMF: a Java open-source data mining library*. Retrieved on October 30, 2016, from http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

[33] Fournier-Viger P, Gomariz Gueniche T A, Soltani A, Wu C and Tseng V S 2014 SPMF: a Java open-source pattern mining library. *Journal of Machine Learning Research* 15: 3389–3393