Pattern mining for large distributed dataset: A parallel approach (PMLDD)

Amrit Pal[1*], Manish Kumar[1]
[1]Department of Information Technology, Indian Institute of Information Technology
Allahabad, India
[e-mail: rs167@iiita.ac.in ; manish@iiita.ac.in]
*Corresponding author: Amrit Pal

Received December 8, 2017; accepted June 5, 2018; published November 30, 2018

Abstract

Handling vast amount of data found in large transactional datasets is an obvious challenge for the conventional data mining algorithms. Addressing this challenge, our paper proposes a parallel approach for proper decomposition of mining problem into sub-problems in order to find frequent patterns from these datasets. The proposed, Pattern Mining for Large Distributed Dataset (PMLDD) approach, ensures minimum dependencies as well as minimum communications among sub-problems. It establishes a linear aggregation of the intermediate results so that it can be adapted to large-scale programming models like MapReduce. In this context, an algorithmic structure for MapReduce programming model is presented. PMLDD guarantees an efficient load balancing among the sub-problems by a specific selection criterion. Further, it optimizes the number of required iterations over the dataset for mining frequent patterns as compared to the existing approaches. Finally, we believe that our approach is scalable enough to handle larger datasets in terms of performance evaluation, and the result analysis justifies all these mentioned concerns.

Keywords: Association rule, Frequent pattern, HDFS, Large datasets, MapReduce

http://doi.org/10.3837/tiis.2018.11.007 **ISSN : 1976-7277**

1. Introduction

Mining of frequent patterns/itemsets and finding association among them, is a crucial area in the field of data mining. It can be used to discover the intricate relationship between the items in the large complex structured and unstructured datasets. It has a wide range of applications starting from a simple grocery store to a well convoluted defense system. Today, we are in the era of automation through the new technologies which generate a colossal amount of data and this is increasing exponentially. Gradually, this extensive amount of data is creating several new challenges for the available mining algorithms. When this enormous amount of data reaches to that extent that it becomes very difficult to manage using traditional database management system, then it is known as Big data [1]. Social networks like Facebook, Twitter, etc. are generating petabytes of data and the human generated data will reach about 44 zettabytes by 2020; certainly managing this huge amount of data is a tough challenge for the researchers. The nature of this data can be described using a 5V [2] structure: 'V's stand for Volume, Velocity, Variety, Veracity, and Value. These Vs also impose challenges to the Big data management and information retrieval [3]. Distributed approaches provide scalable solutions for retrieval of relevant information from this data.

These approaches work through storing data over a distributed storage by dividing it into small chunks/blocks and apply distributed processing for information retrieval from this data. Association mining has two intrinsic tasks: frequent itemset generation which involves frequency distribution calculation about each data item/object, and relationship estimation among them. Sequential approaches require a centralized availability of data to apply an iterative model for association mining problem [4,5,6]. These approaches lack in terms of scalability and have high I/O and communication cost. Thus, these are not sufficient to deal with today's Big data. Remodeling of these approaches and reducing strong dependencies [7] is a complex and comprehensive task. Hence, a convenient remodeling of available frequent pattern mining algorithms according to the comprehensive approaches is urgently required.

A comprehensive approach, in this context, would be the distribution of one complete problem over a cluster where sub-tasks independently run on each node of the cluster and results from nodes are combined to generate the final result. This process may increase the performance and scalability of overall mining process. However, it raises some inescapable issues like inter process communication overhead, central node management, failure management, workload distribution, and mixing the results from the sub-problems.

There are distributed solutions available for processing and storing data like MapReduce [8], Spark, [9] and Hadoop Distributed File System (HDFS) [10]. The use of MapReduce provides independence from memory constraints which in addition gives a prodigious amount of scalability. This MapReduce approach is used in several techniques for association mining. The major problems which exist while remodeling the sequential association mining algorithms are as follows:

1) Several iterations of the MapReduce phase, which results in increasing communication cost, I/O, and time required for overall approach.
2) A proper independence among the sub-problems is required for a scalable implementation of the distributed approach.
3) An appropriate aggregation of the intermediate results.

Addressing these issues our proposed synthesized approach considers the distribution of the data over a cluster. Through this approach:

1) Improvement in the iterative modeling of the MapReduce is done.
2) Proper independence is maintained using distributed parallel counting and tree generation algorithm for generation of FP-Tree.
3) The proposed approach requires no communication between the mappers, also there is no data exchange between the nodes of the cluster.
4) A proper mixture model for intermediate output aggregation is proposed for the generation of complete result.

Rest of the paper is organized as, section 2 discusses about the FP Growth problem, section 3 provides the relevant work done in the mentioned area, section 4 defines the problem statement, section 5 elaborates the process of distributed frequent pattern mining, section 6 explains the specific problem and solution with an example, section 7 demonstrates the performance evaluation and presents the experimental analysis of the proposed approach, and finally section 8 concludes the work.

2. FP-Growth

FP-Growth algorithm is a well-known algorithm for mining frequent itemsets and generation of association rules [12]. This algorithm completely scans the dataset twice: first, it generates the count for each item and second, it produces a tree which is known as FP-Tree (Frequent Pattern Tree). Through Recursive mining on this tree, frequent itemsets and association rules can be generated. Pseudo code for FP-Growth can be understood from algorithm 1. For a transactional dataset, each transaction is added to the FP-Tree, where, in case of first transaction, it is added to the tree as a branch. All subsequent transactions are then added with matching the prefixes. Until the prefix is matched, count of the nodes in the tree for that particular transaction is increased by one and the rest of the transactions is further added as a branch from the point of unmatched prefixes.

```
Algorithm 1  FP-Growth
Required $D_i$
Define a counter array count []
For each transaction j in D
    For each item k in transaction j
        $count[k]=count[k]+1$
    EndFor
EndFor
Sort the transactions according to count []
Remove the infrequent items based on minimum support value.
For each transaction j in D
    add $j^{th}$ transaction to FP-Tree
EndFor
```

There are some issues while mining large amount of data using FP-Growth algorithm:

1) **Volume:** As the load of data is increasing, the size of the transactional database is also increasing. Therefore, FP-Growth needs to be modified to overcome this challenge.

2) **Complexity:** The big chunks of data can be divided into different shards and processed independently to reduce the computational complexity. This is also needed to be done in case of frequent itemset mining. So, parallel implementation of the FP-Growth can result in less complexity.
3) **Iterative approach:** There is a need of reduction in the number of iterations required for generation of frequent itemset and association rules.

The overall mining problem requires distributed count based sorting in order to maintain the dependency among the items. The FP-Growth algorithm follows a sequential approach and to remodel this algorithm in a distributed manner, a division of the problem into sub-problems is highly needed. Here, an effective intermediate result storage mechanism is required for minimizing overall communication overhead. Modeling of the algorithm into MapReduce programming model is also necessary to increase the scalability of overall mining process. To efficiently complete this remodeling process, a < *key; value* > based result generation and mixture is required. In the end, a proper aggregation of the results is essential so that the combined information could have the same accuracy as that of a centralized approach.

3. Related Work

The problem of finding frequent itemsets and discovering the significant association rules has been widely studied in different areas related to data mining. Two of the most prominent algorithms in this field are Apriori [11] and FP-Growth [12], which conventionally follow a sequential approach for mining frequent itemset and association rule. There is an immense need to remodel these algorithms in order to handle the massive amount of dataset problem in a distributed manner and also to adapt distributed programming model like MapReduce. Several implementations of Apriori have already been done; each of which goes through a number of iterations of the MapReduce task as long as the size of frequent itemset [13,14,15,16,17,18]. FP-Growth is also used with MapReduce [19,20] and Spark [21] for addressing the same concerned problem. Again, there are the cases where a combination of Apriori and FP-Growth [22] leads to better results. Tsay, Y. J et al. has proposed Frequent Items Ultrametric Tree (FIUT) algorithm [23] which follows a bottom-up approach as compared to FP-Growth algorithm and requires a lexicographic order to be maintained among the items. This algorithm first extracts the *k-frequent* itemsets which is same as in case of Apriori algorithm [11]. Then these *k-frequent* itemsets are recursively used for finding frequent itemsets of size less than *k* using the ultrametric trees. Zahra Farzanyar et al. has proposed Improved MapReduce Apriori Algorithm (IMRApriori) [24] which uses Apriori as base algorithm and extracts the frequent itemsets from the dataset. This algorithm performs a voting over the mappers in order to distinguish between evident frequent itemset and infrequent itemset. Another approach for mining frequent itemsets using MapReduce has been proposed by Haoyuan Li et al. in Parallel FP-Growth (PFP) algorithm [25] which prepares two lists: one for count maintenance and other for the grouping of 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. To overcome these issues Jiayi Zhou et al. has developed an algorithm, Tidset-Based Parallel FP-Tree (TPFP) [26], which shares the transaction index list (tid list) that efficiently reduces the amount of data exchange. Matteo Riondato et al. has implemented an approach PARMA [27]; orthogonal to the TPFP and PFP, which uses random sampling of the data and applies FP-Growth to produce the approximate results. Further, in case of continuous updating of data, new data can be added monotonously to fulfil these requirements through PIFP algorithm [28], suggested by Xiaoting Wei et al. that performs continuous update to the FP-Tree using a revised threshold value. Sandy Moens et al. has proposed two approaches: first dist-Eclact and further its optimized version BiGFIM [29] for implementation over large datasets. In BiGFIM algorithm, construction of *k-size* frequent itemset is done in parallel using dist-Eclat, which are then used to create *k-size* prefix trees. These prefix trees are distributed to the workers (processing units in a cluster) for further identification of frequent itemsets independently from local data. This algorithm considers the frequency ordering of the dataset and also requires several iterations for finding *k-size* frequent itemset. The decision of the *k* value affects the overall performance of the algorithm. BiGFIM provides better performance than other algorithms like PFP, which is highly adopted and compared in the literature.

Yaling Xun et al. has proposed FiDoop algorithm [30] that uses FIUT algorithm and MapReduce. FiDooP takes only three iterations of MapReduce where first two iterations are for 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. Then, trees having the same number of items, are assigned to single reducer for further decomposition or information extraction. The algorithm requires itemset ordering which is not always present in the real time data [35,36]. Load balancing is required in the third iteration, as it includes the decomposition process. It is recursive in nature and can result in uneven distribution of load on nodes executing mappers. Although algorithm shows better performance than the PFP algorithm which requires multiple iteration of MapReduce. But 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. Based on the literature discussed so far some problems have been identified with existing approaches that require immediate attention:

1) There are problems with inter process communication between mappers.
2) Producing the exact results as the sequential approach is also very troublesome.
3) Number of MapReduce iterations that are required to complete the mining process.
4) The management of the intermediate results.

4. Distributed Frequent Pattern Mining Problem

Mining of association rules and frequent pattern mining are cohesively bonded together. There are several application domains like market basket analysis, sensor networks [36] where mining of frequent itemset is essential for finding dependence among items (even in case of sensors). To make out the complete process of mining frequent itemsets and identification of the association rules, the following terms need to be considered.

1) **Item {i}:** It is a unique entity in a dataset.
2) **Itemset {I}:** A set of non-repeating items $I=\{i_1,i_2,\ldots, i_n\}$.
3) **Transaction {T}:** A set of any permutation of non-repeating items.
4) **Transactional Dataset {D}:** It is a collection of transactions.
5) **Support Sup:** The support for a pattern $P\subset I$ is the number of times P has occurred in the entire database.
6) **Frequent Itemset pattern:** An Itemset is a frequent itemset if it occurs in a given database more than a threshold value. This threshold value is known as minimum support.

7) **Confidence:** Confidence is calculated among two itemsets e.g. I_1, I_2 in context of co-occurrence among them. Confidence of an association rule $I_1 \rightarrow I_2$ is given by the equation below.

$$conf\,(I_1 \rightarrow I_2) = \frac{\text{supp}(I_1 \cup I_2)}{\text{supp}(I_1)} \qquad (1)$$

A broad scenario regarding our problem formulation is to find all frequent patterns/itemsets from a given dataset based on a minimum support value so that association rules based on confidence values can be generated for these itemsets. A centralized approach considers all data to be available in one big chunk for iterative scanning of the data. While considering the challenges of Big data and its storage, distributed systems can provide a good solution as the data is distributed over a cluster. Consider a transactional dataset *D*, distributed over a distributed storage as $D_1, D_2, \ldots, D_k$. Basic methodology behind the calculation of association rules is the conditional probability computation among the itemsets. Performing the same operation over a distributed environment requires different approaches, as the overall sample space and the entire data is partitioned across a network. The complete problem can be summerized as follows:

1) Distribution of the scanning process over the complete dataset such that the results are same as these are in a centralized approach.
2) Remodeling of itemset generation process in distributed manner and aggregation of the results.
3) To find the frequent itemsets independently from the distributed dataset and aggregation of the results for global frequent itemsets.
4) Finally, generation of the complete synthesized information which can be used for association rule generation.

5. Pattern Mining for Large Distributed Datasets (PMLDD)

The overall proposed approach of mining frequent itemsets and production of association rules is divided into two phases of MapReduce as shown in **Fig. 1**. There are exactly two iterations of MapReduce required to complete the proposed approach. Scanning the data includes transaction by transaction scan on the available input dataset which is done by the mappers and further mappers generate all the intermediate results which are aggregated to achieve the final result by the reducers. Mappers, in the first phase, form a < *key; value* > pair where, each key corresponds to the *item* occurred in the transactions and the value refers to its *count*. Different mappers produce these < *key; value* > pairs in each transaction and pass these for further processing to the reducers.

5.1. Phase 1

First phase includes one complete scan of input data, one iteration of mapper and reducer to complete this phase. Mapper and reducer strictly follow the MapReduce classical model for extracting the local information from available data and linearly combining that information.

5.1.1 Mapper function 1

Each mapper starts with scanning its data part D_i and generates the intermediate output. Mapper function has a conventional application of counting which is very fast in case of distributed systems. Each mapper takes part in the counting process by notifying the existence of an *item* as < *key; value* > pair in form of < *item; 1* > as given in the algorithm 2. The resulting pair is written to HDFS for its availability for next stage of phase 1

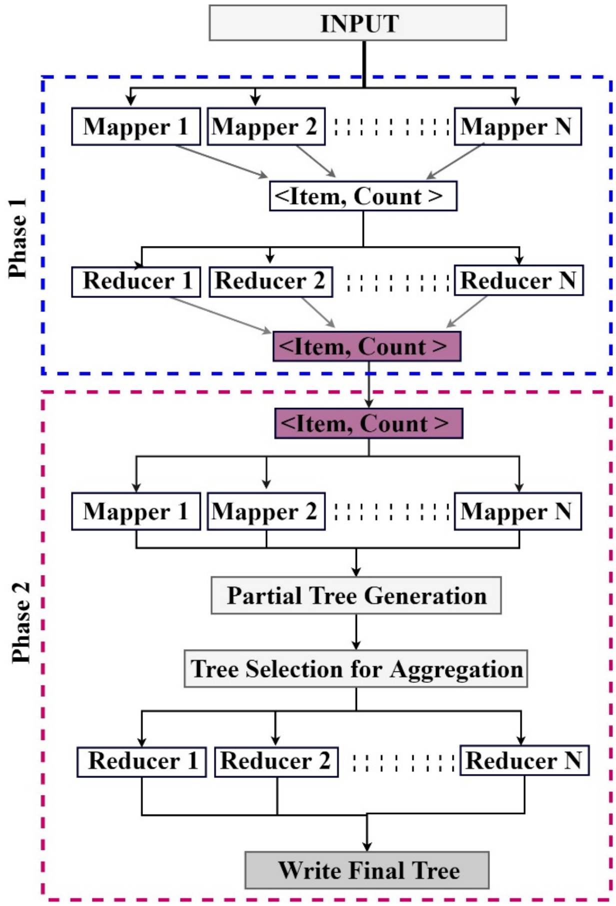

Fig. 1. Two phases of proposed approach

Algorithm 2 Mapper 1

```
Required $D_i$
For each item in $T_j \in D_i$
    write<item, 1>
EndFor
write<item,1>
Return<item, 1>
```

Algorithm 3 Reducer 1

```
Required <item, 1>
While new item do
    count all different items
EndWhile
write<item, count>
Return<item, count>
```

5.1.2 Reducer function 1

In its first scan, the reducer takes the pairs generated by the mappers as input and combines these pairs to get the global count for the items by summation over each < *key; value* > pair based on key's value. This global information is stored in the distributed storage pool to make it available to the mappers for the next phase. Algorithm 3 describes the complete phase 1 reducer execution.

5.2 Phase 2

Even in the second phase of the proposed approach, one iteration of mapper and one iteration of reducer is required. Each mapper builds the conditional FP-Tree from the available dataset and the reducers combine these FP-Trees using the tree aggregation process.

5.2.1 Map Function 2

From the first phase, the frequency information for items is extracted. Each mapper in phase 2 then starts scanning the transactions and performs sorting the transactions based on the information gathered from first phase and eliminates the infrequent items based on the minimum support value. Mappers use the sorted transactions for creating the conditional tree (FP-Tree) based on the occurrence of an item in a particular transaction. Each mapper constructs a local tree and writes these trees to the files over the distributed storage pool. Different mappers write different trees; these trees have the compressed information about the items and transactions in the dataset. The process of formation of trees can be understood from the pseudo code in algorithm 4 and 5.

1) The referred algorithm 4 takes the subsets of primary dataset and the global count list as the input and creates a null tree.
2) It selects a transaction, sort it according to the $Count_{List}$ and removes infrequent items.
3) It uses *insert* function as described in algorithm 5 for inserting a transaction into the tree.
4) Repeats the process until all transactions from dataset D_i are added to the tree.

At first, there is a null tree for each mapper in which mappers add transactions one by one to produce FP-Tree. In the transaction insertion process, each transaction's prefixed item is checked for the current children of the considered node using 4th statement of algorithm 5. If no similar child is found in the tree, then the complete transaction is added as a new branch in the tree with count of each item is set as one. If there is a similar child as the considered prefixed item, then the count for the item is increased by one. And recursive call to the function is made considering next item in the transaction and child of the currently selected node. This process is iteratively executed until there are transactions available in the dataset.

```
Algorithm 4 Mapper 2
Required $D_i$ , $Count_{List}$
1. create a null tree Tree
2. For each $T_j \epsilon D_i$ do
3.     sort transactions according to the count
4.     remove infrequent items from $T_j \rightarrow T_s$
5. insert($T_s$, Tree)
6. EndFor
7. write Tree
8. Return Tree
```

```
Algorithm 5 insert(T,Tree)
Required T,Tree
1. insert(t|T,Tree)
2. T=t|T-t
3. root= Root(Tree)
4. $child_m$= Child(root)
5. If $child_m$.name=t.name
6.     $child_m$.count=$child_m$.count+1
7.     insert(T, $child_m$)
8.     Return
9. EndIf
10. Child(root)=t|T
11. Set count for each item to 1
12. Return Tree
```

5.2.2 Reducer Function 2

In the second phase, reducer performs the aggregation of the trees by selecting any two trees from the distributed storage. Reducer then combines these trees in a level-wise manner and stores the newly generated tree again into the distributed storage. The selection of the trees from the distributed storage pool is done iteratively until there is more than one tree in the distributed storage; the overall merging works as follows:

1) Select two trees from the distributed storage as described in algorithm 6.
2) Merge these trees using merge function as described in the pseudo code of algorithm 7.
3) Root node of each tree is selected.
4) Each *node* of the second tree is compared with the child of the first tree's *node*.
5) If any match is found, then count of that node is increased by adding the count of both the nodes and recursive call to *merge()* is done by making the child nodes as the root nodes.

```
Algorithm 6 Reducer 2
Required Dist_Tree_Pool
While {number of trees in Dist_Tree_Pool > 1} do
    Select any two trees from Dist_Tree_Pool
    Select root node of the trees as node1 and node2
    merge (node1,node2)
    write the resultant tree to distribute storage
EndWhile
```

```
Algorithm 7 Merge
Required $Tree_i$, $Tree_j$
$node_i=root_i$
$node_j=root_j$
For each $child_m$ of $node_j$ do
    If $child_m$ is also child of $node_i$
        find corresponding $child_n$ from $node_i$
        $merge(child_n, child_m)$
        $child_n.count=child_n.count + child_m.count$
    EndIf
    Else add $child_m$ to $node_i$ as another child
EndFor
Return Tree
```

The complete two phased proposed approach ends with a single tree left in the distributed storage. Each *write* operation in the algorithms can be of two types, first, we can manage the trees in memory or second, we can use the Newick Tree [32] structure for storing these trees to files. Newick Tree format is a widely used representation of the trees and a scalable implementation is available as ETE toolkit [33] which is widely used for GNOME analysis and visualization [37]. The tree can be stored as a tree object over distributed storage with each node's count information. This final tree has the sketched information about the whole input data. The resultant tree can be used for mining the frequent itemsets and further for association rules mining by simply traversing over the tree.

6. Example Problem

Simply applying FP-growth algorithm and discovering association rules can be understood using the examples in the literature [12]. To realize the process of mining the frequent itemsets and computation of association rules using proposed approach, consider the example dataset as detailed in **Table 1**. In this context, we explain the process of local tree generation and combining the intermediate trees to build the complete tree. In the *first phase*, division of the complete dataset takes place as shown in tables D_1, D_2, D_3. Then each mapper scans the transactions of the subset dataset and generates the key value combinations. These combinations are then processed by the reducers to estimate the global counts. This information is shared using the distributed storage and made available for second phase of the mappers and reducers. In the *second phase*, each mapper sort the transactions and starts generating the local FP-Tree. Considering the minimum support to *30%*, the items I_1, I_6, I_7, I_8 got ignored. After that the sorting by each mapper is done independently for each transaction in the subset dataset.

Table 1. Transaction Dataset D and its division D_1, D_2, D_3

D

Transaction ID	Transaction
T01	I_1 I_2 I_4
T02	I_3 I_2 I_5
T03	I_4 I_5 I_9 I_6
T04	I_4 I_9 I_{11} I_7
T05	I_2 I_{11}
T06	I_2 I_8

D1

Transaction ID	Transaction
T01	I_1 I_2 I_4
T02	I_3 I_2 I_5

D2

Transaction ID	Transaction
T03	I_4 I_5 I_9 I_6
T04	I_4 I_9 I_{11} I_7

D3

Transaction ID	Transaction
T05	I_2 I_{11}
T06	I_2 I_8

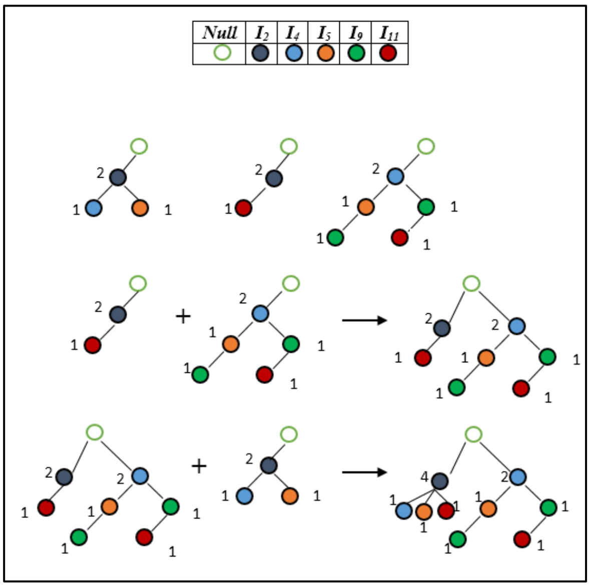

Fig. 2. Tree combining process

Each mapper starts forming the partial trees and stores these trees in the HDFS. Three independent trees by three mappers are created in our example problem. During second phase, the reducers perform the aggregation of trees by selecting two trees at a time from distributed pool. Then carry out the merging operation on those trees as shown in **Fig. 2**. When there is no tree left for merging, the final tree is created and can be used for finding the association rules by analyzing the tree.

7. Performance Evaluation and Discussion

7.1 Experimental Setup

Proposed approach requires a distributed storage and data processing environment; for this, a five node Hadoop cluster has been setup to validate the proposed approach. There are many physical and logical modules in a Hadoop cluster, but Namenode, Datanode, and Secondary Namenode are its main components. The cluster follows a star topological architecture where nodes are connected through a switch directly with each other, each node has; Ubuntu 16.04 as operating system, Intel(R) Core(TM)*i3*-32320 processors, 6 GB of RAM and Hadoop 2.6.4 installed on it. The number of mappers is 4 and replication factor is set to 2. Our proposed parallel approach is applied using MapReduce. Each mapper independently executes according to the proposed approach and produces the results which are further combined by the reducer functions.

7.2 Analysis

Generation of FP-Tree requires insertion of each pruned transaction in the tree. The overall time required to add a new transaction *T* in FP-Tree is *O(|freq(T)|)*, where *freq(T)* is the set of frequent items. For the complexity analysis of the proposed approach, we consider the number of items is *N*, maximum length of any transaction is *l*, number of mappers is *k*, number of data partitions is *p*. Then the time complexity of the entire approach can be estimated as $O(l * \frac{p}{k} * N\ log\ N)$. Further analysis of the proposed approach is done in the following manner:

1) The time required by different phases of the proposed approach to process the data.
2) The percentage time spent in each phase of the algorithm.
3) Visualization of the proposed algorithm and the intermediate results.
4) Analysis of the proposed approach with the existing approaches.
5) Scalability analysis of the proposed approach.
6) Analysis based on the tree selection policy for aggregation.

The datasets considered in these analyses are Mushroom [31] dataset and IBM synthetic dataset generator [34], which has been used in previous works [29]. Firstly, we have used Mushroom dataset for 1,2, and 3, which contains *119* number of items and *8124* number of transactions. **Fig. 3** shows the time required by different stages of proposed approach. This analysis is done by varying the support value, so that, size and number of trees generated can be varied.

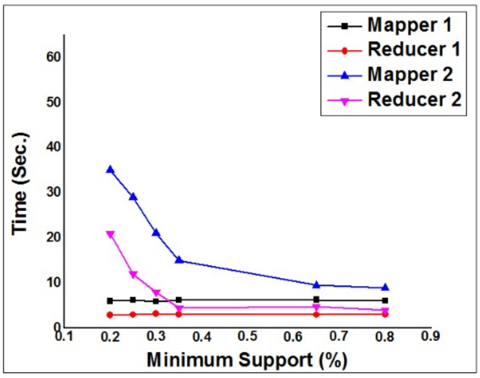

Fig. 3. Time required by each phase of PMLDD based on different support values

The amount of time required by the mappers and reducers to complete the first phase is almost constant for varying values of the minimum support. But the time required by the second phase of the algorithm is high for low values of minimum support due to the increased levels and degree of nodes in the tree. To visualize these trees, we have used ETE toolkit [32,33] which is highly used for GNOME analysis. The overall scenario of the aggregation process can be visualized using **Fig. 4**(a, b, c, d), which shows the projection of intermediate FP-Trees. Each node in the tree represents an item as shown in **Fig. 5** (circled area of **Fig. 4(a)**). The aggregation process results in a complete tree as shown in **Fig. 4(e)**

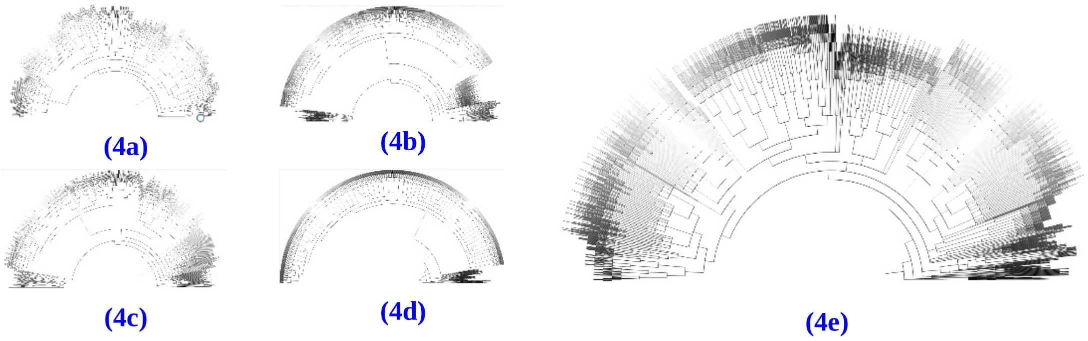
(4a)
(4b)
(4c)
(4d)
(4e)

Fig. 4. Visualization of the aggregation process for Mushroom dataset using PMLDD approach

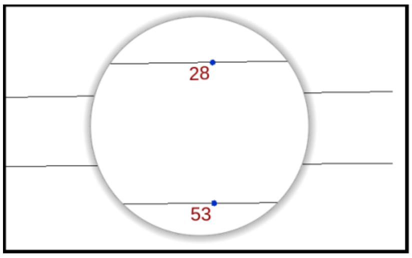

Fig. 5. Edges of the Mushroom 1 tree

Further analysis for scalability and comparison with the existing approaches has been done using synthetic dataset which contains *1000* number of items and *1,000,000* number of transactions. We have analyzed the percentage time spent by each mapper and reducer in each phase; **Fig. 6** shows this analysis. Maximum amount of time required by the proposed algorithm is in the conditional tree generation and aggregation process. Based on the varying value of the support count, the number of trees generated by the mappers varies significantly. It affects the amount of time required by the reducers to process the aggregation process. The ratio of the time required by the different stages of the algorithm is analyzed as shown in the **Fig. 6**.We have compared the proposed approach with the available approaches. The proposed approach gives a significant performance improvement as shown in **Fig. 7**.

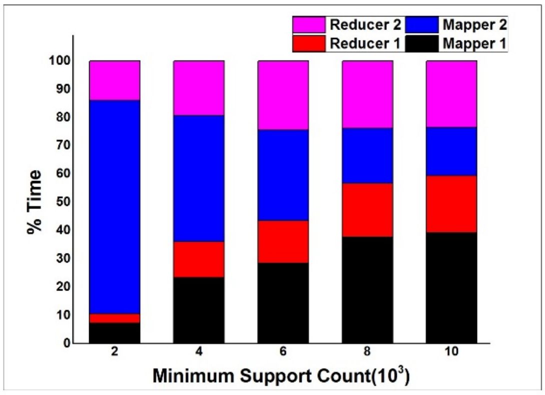

Fig. 6. Percentage of time spent by PMLDD in each phase of the MapReduce

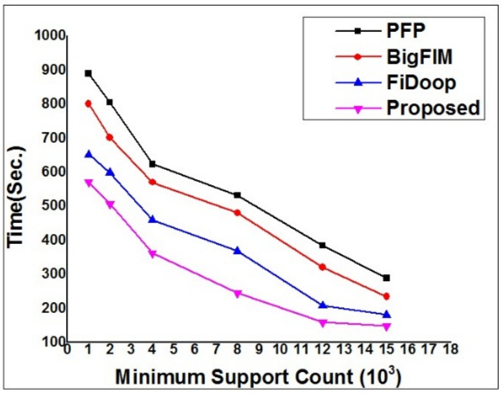

Fig. 7. Proposed approach compared with the existing approaches based on the minimum support values scaled by 10^3

7.3 Scalability Analysis

Scalability analysis has been done for performance evaluation using a scalability factor S_{Fact}using equation 2.

$$S_{Fact} = \frac{F_T}{C_T} \qquad (2)$$

F_T is the time required in first case in which only two cores are available for mining process. This time will act as the base for comparison. C_T is the time required in all subsequent cases when we continuously increase the number of available cores. An ideal scenario is that the S_{Fact} should be increased in the same ratio as the number of cores are increasing. The actual scenario is different from ideal as shown in **Fig. 8**. It shows the analysis of the complete process with varying number of cores. Synthetic dataset has been used for this process and ten million transactions have been used for testing the scalability of the system. Analysis shows that a good amount of scalability can be achieved using the proposed approach.

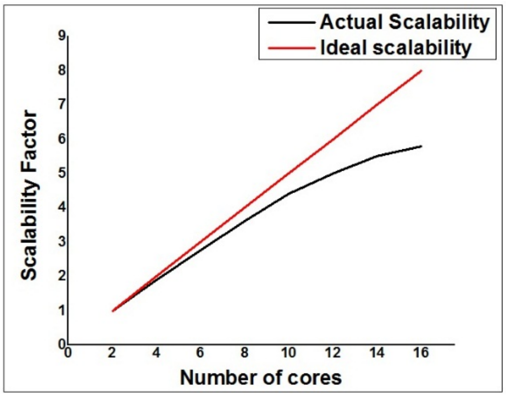

Fig. 8. Scalability analysis of PMLDD with increasing number of cores in the cluster

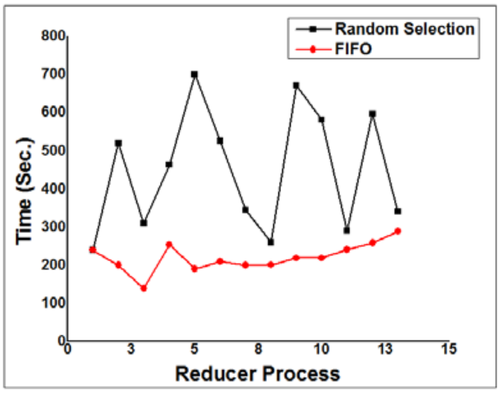

Fig. 9. Tree selection process analysis of different reducer processes in tree aggregation

7.4 Load Balancing

Aggregation process analysis has been done using two tree selection criteria. Selection of the trees for aggregation can drastically affect the performance of the mining process. The compressed trees generated by the mappers are of varying heights and sizes. The selection of trees can be of two types: random selection of trees and First In First Out (FIFO). These selections result in different processing time required by the reducers to merge these trees. As shown in **Fig. 9**, the amount of time required by the reducers for completing the aggregation process follows a zigzag shape for random selection of the trees and some better smooth shape in case of FIFO. This analysis considers the average time required by the reducers to perform the aggregation process. The *x-axis* shows the reducer process and *y-axis* shows the time required by that reducer for tree aggregation.

The overall analysis shows that proposed approach works well over the distributed storage. It shows significant performance improvement over the existing approaches.

8. Conclusion

This paper presents a PMLDD approach in order to deal with the inherent problem of data mining in terms of managing large transactional datasets and finding frequent patterns. While ensuring minimum dependence among the sub-problems, the overall approach requires two MapReduce iterations and hence reduces the database scanning load over the distributed storage. An efficient result aggregation scheme is presented and tested over different datasets. The method is ample scalable to be adapted for distributed programming model and to achieve the scalability in result aggregation phase, a linear aggregation of the results is done. Obtained results validate the efficacy of PMLDD concept in terms of reduction in the amount of time with existing approaches. In future, we would like to extend our work through its application level implementation into the emerging technology like Internet of Things (IoT) for event based pattern mining.

References

[1] Wu, X., Zhu, X., Wu, G. Q., & Ding, W., “Data mining with big data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97-107, Jan. 2014. Article (CrossRef Link)

[2] Fan, W., Bifet, A., “Mining Big Data: current status, and forecast to the future,” *ACM SIGKDD Explorations Newsletter*, Vol. 14, no. 2, pp. 1-5, Apr. 2013.Article (CrossRef Link)

[3] Rodrguez-Mazahua, L., Rodrguez-Enrquez, C. A., Snchez-Cervantes, J. L., Cervantes, J., Garca-Alcaraz, J. L., & Alor-Hernndez, G., “A general perspective of Big Data: applications, tools, challenges and trends,” *The Journal of Supercomputing*, Vol. 72, no. 8, pp. 3073-3113, Aug. 2016.Article (CrossRef Link)

[4] Hipp, J., Gntzer, U., & Nakhaeizadeh, G., “Algorithms for association rule mining a general survey and comparison,” *ACM Sigkdd Explorations Newsletter*, Vol. 2, no. 1, pp. 58-64, Jun. 2000. Article (CrossRef Link)

[5] Han, Jiawei, Jian Pei, and Micheline Kamber. “Data mining concepts and techniques,” *Elsevier*, 2011. Article (CrossRef Link)

[6] Seol, W. S., Jeong, H. W., Lee, B., & Youn, H.Y., “Reduction of Association Rules for Big Data Sets in Socially-Aware Computing,” in *Proc. of Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE)*, pp. 949-956, Dec. 2013. Article (CrossRef Link)

[7] Del Ro, S., Lpez, V., Bentez, J. M., & Herrera, F., "A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules," *International Journal of Computational Intelligence Systems*, Vol. 8, no.3, pp. 422-437, May. 2015. Article (CrossRef Link)

[8] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, Vol. 51, no. 1, pp. 107-113 Jan. 2008.Article (CrossRef Link)

[9] Apache Spark, accessed 15 Jun. 2017. Article (CrossRef Link)

[10] Shvachko, K., Kuang, H., Radia, S., & Chansler,R., "The hadoop distributed file system," in *Proc. of Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1-10, May. 2010.Article (CrossRef Link)

[11] Agrawal, R., Imieliski, T., & Swami, A., "Mining association rules between sets of items in large databases," in *Proc. of Proceedings of the ACM SIGMOD*, Vol. 22, no. 2, pp. 207-216, Jun. 1993. Article (CrossRef Link)

[12] Han, J., Pei, J., Yin, Y., & Mao, R., "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, Vol. 8, no. 1, pp. 53-87, Jan. 2004. Article (CrossRef Link)

[13] Lin, M. Y., Lee, P. Y., & Hsueh, S. C., "Apriori based frequent itemset mining algorithms on MapReduce," in *Proc. of Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, pp. 76, Feb. 2012. Article (CrossRef Link)

[14] Yang, X. Y., Liu, Z., & Fu, Y., "MapReduce as a programming model for association rules algorithm on Hadoop," in *Proc. of Proceedings of the 3rd International Conference on Information Sciences and Interaction Sciences (ICIS)*, pp. 99-102, Jun. 2010. Article (CrossRef Link)

[15] Chang, X. Z., "MapReduce-Apriori algorithm under cloud computing environment," in *Proc. of Proceedings of the International Conference on In Machine Learning and Cybernetics (ICMLC)*, Vol. 2, pp. 637-641, Jul. 2015. Article (CrossRef Link)

[16] Lin, X., "Mr-apriori: Association rules algorithm based on MapReduce," in *Proc. of Proceedings of the 5th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 141-144, Jun. 2014. Article (CrossRef Link)

[17] Li, N., Zeng, L., He, Q., & Shi, Z., "Parallel implementation of Apriori algorithm based onMapReduce," in *Proc. of proceeding of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD)*, pp. 236-241, Aug. 2012. Article (CrossRef Link)

[18] Guo, J., & Ren, Y. G., "Research on Improved Apriori Algorithm Based on Coding and MapReduce," in *Proc. of proceeding of the IEEE 10th International Conference on Web Information System and Application Conference (WISA)*, pp. 294-299, Nov. 2013. Article (CrossRef Link)

[19] Chang, H.-Y.; Lin, J.-C.; Cheng, M.-L. & Huang, S.-C. "A Novel Incremental Data MiningAlgorithm Based on FP-growth for Big Data," in *Proc. of proceeding of the International Conference on Networking and Network Applications (NaNA)*, pp. 375-378, Jul. 2016. Article (CrossRef Link)

[20] Chang, H.-Y.; Tzang, Y.-J.; Lin, J.-C.; Hong, Z.-H.; Chi, T.-Y. & Huang, C.-Y., "A Hybrid Algorithm for Frequent Pattern Mining Using MapReduce Framework," in *Proc. of proceeding of the First International Conference on Computational Intelligence Theory, Systems and Applications (CCITSA)*, 19-2, Dec. 2015. Article (CrossRef Link)

[21] Deng, L. & Lou, Y. "Improvement and research of FP-growth algorithm based on distributed spark," in *Proc. of proceeding of the International Conference on Cloud Computing and Big Data (CCBD)*, 105-108, Nov. 2015. Article (CrossRef Link)

[22] Lan, Q.; Zhang, D. & Wu, B. "A new algorithm for frequent itemsets mining based on apriori and FP-Tree," in *Proc. of proceeding of WRI Global Congress on Intelligent Systems*, pp.360-364, May. 2009. Article (CrossRef Link)

[23] Tsay, Y. J., Hsu, T. J., & Yu, J. R., "FIUT: A new method for mining frequent itemsets," *Information Sciences*, Vol. 179, no. 11, pp. 1724-1737, May. 2009. Article (CrossRef Link)

[24] Farzanyar, Z. & Cercone, N. "Efficient mining of frequent itemsets in social network data based on MapReduce framework," in *Proc. of proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 1183-1188, Aug. 2013. Article (CrossRef Link)

[25] Li, H.; Wang, Y.; Zhang, D.; Zhang, M. & Chang, E. Y. "Pfp: parallel fp-growth for query recommendation," in *Proc. of proceeding of the ACM conference on Recommender systems*, pp. 107-114, Oct. 2008. Article (CrossRef Link)

[26] Zhou, J. & Yu, K.-M. "Tidset-based parallel FP-tree algorithm for the frequent pattern mining problem on PC clusters," in *Proc. of proceeding of the International Conference on Grid and Pervasive Computing*, pp. 18-28, 2008. Article (CrossRef Link)

[27] Riondato, M.; DeBrabant, J. A.; Fonseca, R. & Upfal, E. "PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce," in *Proc. of proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 85-94, Oct. 2012. Article (CrossRef Link)

[28] Wei, X., Ma, Y., Zhang, F., Liu, M., & Shen, W., "Incremental FP-Growth mining strategy for dynamic threshold value and database based on MapReduce," in *Proc. of proceedings of the IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 271-276, May. 2014. Article (CrossRef Link)

[29] Moens, S., Aksehirli, E., & Goethals, B., "Frequent itemset mining for big data," in *Proc. of proceeding of the IEEE International Conference on Big Data*, pp. 111-118, Oct. 2013. Article (CrossRef Link)

[30] Xun, Y., Zhang, J., & Qin, X., Fidoop: "Parallel mining of frequent itemsets using mapreduce," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 46, no. 3, pp. 313-325, Mar. 2016. Article (CrossRef Link)

[31] M. Lichman "UCI Machine Learning Repository," *University of California, Irvine, School of Information and Computer Sciences*, 2013. Article (CrossRef Link)

[32] Huerta-Cepas, J., Dopazo, J., & Gabaldn, T., "ETE: a python Environment for Tree Exploration," *BMC bioinformatics*, Vol. 11, no. 1, pp.24, Jan. 2010.Article (CrossRef Link)

[33] Huerta-Cepas, J., Serra, F., & Bork, P., "ETE3: Reconstruction, analysis, and visualization of phylogenomic data," *Molecular biology and evolution*, Vol. 33, no. 6, pp. 1635-1638, Feb. 2016. Article (CrossRef Link)

[34] Fournier-Viger, P., Gomariz, Gueniche, T., A., Soltani, A., Wu., C., Tseng, V. S., "SPMF: a Java Open-Source Pattern Mining Library," *Journal of Machine Learning Research (JMLR)*, Vol. 15, PP. 3389-3393, Jan. 2014. Article (CrossRef Link)

[35] Holt, J. D., & Chung, S. M., "Parallel mining of association rules from text databases," *The Journal of Supercomputing*, Vol. 39, no. 3, pp. 273-299, Mar. 2007.Article (CrossRef Link)

[36] Rashid, M. M.; Gondal, I.; Kamruzzaman, J.; "Dependable large scale behavioral patterns mining from sensor data using Hadoop platform," *Information Sciences*, Vol. 379, pp. 128-145, Feb. 2017. Article (CrossRef Link)

[37] Jarvis, E. D., Mirarab, S., Aberer, A. J., Li, B., Houde, P., Li, C., & Suh, A., "Whole-genome analyses resolve early branches in the tree of life of modern birds," *Science*, Vol. 346, no. 6215, pp. 1320-1331, Dec. 2014. Article (CrossRef Link)

Amrit Pal is pursuing his PhD degree from Department of Information Technology at Indian Institute of Information Technology, Allahabad, India. He has received his M.Tech degree from National Institute of Technical Teachers' Training and Research, Bhopal, India in 2014. His research interest includes data mining and Big Data analytics. He has publications in the similar areas.

Manish Kumar has received his PhD degree on Data Management in Wireless Sensor Networks from Indian Institute of Information Technology, Allahabad India in 2011. He received his M.Tech degree in Computer Science from Birla Institute of Technology, Mesra (Ranchi) India. He is a professional member of IEEE and ACM. Currently he is working as an Assistant Professor in Department of Information Technology at Indian Institute of Information Technology, Allahabad India. His research interest includes data mining and warehousing, data management in wireless sensor networks and Big Data Analytics. He has contributed in a number of books in the same areas and has many national and international publications in renowned journals and conferences