ISSN 2278-3091

Volume 9, No.4, July – August 2020

International Journal of Advanced Trends in Computer Science and Engineering

Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse181942020.pdf

https://doi.org/10.30534/ijatcse/2020/181942020

Distributed Association Rules Mining of Varying Data Partition Size Using Nodesets

Manoj Sethi[1], Dr. Rajni Jindal[2]

[1]Department of CSE, Delhi Technological University, ShahbadDaulatpur, Delhi-110042, manojsethi@dce.ac.in

[2]Department of CSE, Delhi Technological University, ShahbadDaulatpur, Delhi-110042, rajnijindal@dce.ac.in

ABSTRACT

Association rule mining in the distributed database has become an important area of research, where frequent pattern or itemsets are found in a large distributed data of varied sizes stored at multiple sites. In recent years, new efficient data structures have been proposed for the data mining. A new algorithm named as QDFIN(Quick distributed frequent itemset mining using nodeset) is proposed in this paper which uses the efficient nodeset data structureto store the candidate itemsets locally at each site and zero-first technique to balance the load and pruning to reduce the candidate sets. The algorithm is implemented and the speed performance is compared with PFIN and FDM using FP-Growth algorithms. Results shows that the proposed algorithm not only outperform other algorithms on varying size data partition but also on uniform distributed data on 4, 5 and 6 node setups.

Key words: Association Rule, Database, Distributed Mining, frequent itemset, Nodeset

1. INTRODUCTION

The voluminous amount of data is created at their different sites and locations which varies in size. It is not feasible to load all data onto a centralized database for analysis due to resource limitations or policy. With the rising and varying size of databases and the demand of mining patterns from data, there is a need to find a solution to analyse the data as and where it is generated and find interesting and frequent pattern mining in the distributed data. The solution is distributed Association rule mining(DARM)[1]. In DARM, data is stored at different locations and various processors work parallel to provide a fast and efficient results. Distributed data mining finds local frequent patterns at different sites, communicates with other sites and finds the global frequent itemsets. As compared to centralized frequent itemsets mining lesser algorithms have been proposed in literature for DARM. Some of the famous DARM algorithms are AprTidRec[2], Fast Distributed Mining of association rules (FDM)[1], Optimized distributed association rule mining (ODAM)[3] etc.[4] states that FIN algorithm is the most recent algorithm and fast in generating frequent itemsets.

1.1. Data Structures used for Mining

Data Structures play an important role in reducing the computational complexity of an algorithm which makes it better. Different data structures are proposed in the literature and based on these data structures different DARM algorithms are proposed. The popular data structures which are used in data mining are FP-tree used by the FP-Growth[5] algorithm; N-List is a structure like FP-tree[6]and stores information obtained from PPC-tree about the itemsets using preorder, postorder; Nodesets is an efficient data structure[7], which stores only postorder or preorder of nodes in the form of N-info; Trie data structure[8]is a prearranged data structure, also known as radix tree, digital tree or prefix tree. In Trie structure strings are used to store the keys.

This paper presents an efficient technique for mining distributed data of varying size to find association rules QDFIN which uses the novel data structure nodeset using FIN[7] algorithm and also best technique for reducing the candidate sets a new zero-first technique for utilizing the sites capabilities in an effective manner. It combines the advantages of local efficiency, load balancing in highly skewed data and low communication load in finding global frequent itemset in place of only one centralized site.

1.2. Paper Organization

Rest of the paper is divided into 5 more sections. Section2 discusses the related work done in this area. Section 3 describes the basic technologies use in the distributed data mining rule mining. In section 4 new algorithm QDFIN is proposed and explained in detail. Section 5 is experimental evaluation and comparison of the proposed algorithm with some of the existing algorithms. Lastly section 6 describes the conclusion and future scope of the work.

2. RELATED WORK

Many surveys have been done on finding association rules in the database[9], [10].Association rule mining gained popularity with an article published in 1993 by [11] which has been sited more than 22000 times according to google scholar.

The apriori like algorithms are based on anti-monotone property[12] and are used to find the frequent itemsets. These algorithms use a strategy to generate and test candidate set[2]. Two variations[2]AprioriTID and AprioriHybrid are also proposed in literature.AprioriTID[2] does not use the database again for finding the frequency of the items after the first pass. AprioriHybrid[2] is also uses Apriori algorithm in the first pass and then moves to AprioriTID at the end of the pass.In 2000, another algorithm FP-Growth[5]was proposed, where a tree like structure is made after database scan. The tree is used to mine frequent itemsets. An algorithm PPC-Tree[13] is proposed in 2012 which uses new structure N-List structure based on PrePost. N-List is a novel vertical data representation structure and it is originated from a FP-Tree like structure.

Algorithm FIN, Fast mining frequent itemset[7] using structure Nodesets and based on the PrePost algorithm is another algorithm proposed in 2014. Nodeset is also based on PPC-Tree butto store information of each node, it makes Nodeset structure using postorder or preorder of the node. DFIN [14] algorithm uses diffnodesets structure which is also based on structure nodeset. In 2015 [6] proposed an Algorithm based on PPC-Tree structure PrePost$^+$. To discover the frequent itemsets, it sets enumeration search tree using the N-List structure. Structure linear prefix tree[15]composed of array forms was introduced which minimizes the pointers between the nodes. IFIN$^+$ algorithm[16] used multi-physical computational units whereas [17] proposed a shared memory parallelism for improving single machine performance for the frequent itemset mining. A framework DT-DPM (Decomposition Transaction for Distributed Pattern Mining) [18]proposed in literature. It integrates Density-Based Spatial Clustering of Applications and distributed computing represented, CPU multi-cores and Single CPU for solving pattern mining problems. Performance of the algorithm[19] depends on the number of nodes in the distributed mining. Execution time improves with increase of number of nodes or transactions. When number of nodes increases but number of transactions are less, algorithms will take longer execution time[20]. FDM optimised with FP-Growth and DiffSet-mining improves the performance of FDM algorithm. A case study[21] shows that the association mining helps in busting the business and predicting the sales.

2.1.Distributed Data Mining Algorithms

Count Distribution(CD) [22] is another important algorithm where the apriori algorithm is run parallelly. In CD, local support count for each itemset is found. The local count is communicated to each site and then by each site global frequent itemsets are found. AprTidRec algorithm is proposed by [12] in 2011, which is based on apriori algorithm. It has only the joint steps and pruning steps. It generates tidRec, a record structure for each candidate frequent itemset. It has better running time than the apriori algorithm.FDM [1]proposed in 1996 is based on Count Distribution and apriori algorithms. It finds the local large itemsets, communicate them to the respective polling sites to find global count which are used to find the global frequent itemsets. The number of messages exchanged are low, just O(n).

A distributed association rule mining algorithm[23],DMA requires just *O(n)*support count messages exchange for each candidate set generated and candidate sets are also low. ODAM (Optimized Distributed Association Rule Mining) [3] algorithm removes the infrequent global frequent size-1 itemsets after discovery then finds larger frequent itemsets. The focus is on the communication and synchronization issues. PFIN algorithm for mining frequent itemsets is proposed by [24], which decomposes the large problem in small tasks executed in parallel and uses nodeset data structure. It is using a hash-based load balancing strategy for optimize resources. Many algorithms are proposed based on the map-reduced technique in distributed environment. MRPrepost[25]and Prepost[26] algorithms are based on map-reduce where first one gives the processing of prepost algorithms where as second one iscloud implementation and apriori map-reduce. Nadar proposed a new data structure nagNodeset[27]used in the algorithm nagFIN. The algorithm is based on the nodes in the prefix tree. There are some negative associations[28]along with the positive associations that exist in the data which are very significant for the business. . [29] proposed an effective algorithm based on optimized matrix computation for Multi party data computation having different challenges.

3. BASIC TECHOLOGIES

3.1 Distributed Data Association Rule Mining

In this proposed work, focus is on distributed association rule mining (DARM), where data are captured or gathered in distributed manner at different location with varying data partition size. One of the popular algorithms for DARM is the FDM (Fast Distributed Algorithm for mining Distributed Association Rules)[1], which uses Apriori algorithm to generate the local frequent itemsets at each site. Another algorithm, FIN (Fast mining frequent itemsets using Nodesets)[7]proposed a new approach based on the PrePost algorithm which having advantages of apriori algorithm as well asFP-Growth algorithm. In this work, FIN[7] algorithm is used in the FDM algorithm to generate the local frequent itemsets in the distributed database with a new balancing technique zero-first. The problem statement is given below as taken from [1] is defined below:

Let DB is a transaction database with $I = \{i_1, i_2, \dots, i_m\}$ set of items. Transaction T of DB is a set of items where$T \subseteq I$. An itemset$Z \subseteq I$, belongs to T if and only if $Z \subseteq T$. An association rule(AR)is represented[1] as $\Rightarrow Y$, where,$Z \subseteq I$ and $Y \subseteq I$ and$Z \cap Y = \phi$.The AR$Z \Rightarrow Y$ holds in the database with a confidence ‘c’ implies that the probability of a transaction in database containing Z also contains Y is ‘c’. The association rule$Z \Rightarrow Y$ has support ‘s’ in database implies that the probability of a transaction in database contains both Z and Y is ‘s’. The association rules mining is a task to search all the association rules in the database where support is greater than the minimum support threshold value and confidence is greater than the minimum confidence threshold value.

For an itemset Z, support is defined as the percentage of transactions in database containing Z, and its support count, Z.sup, is total number of transactions in database containing Z. An itemset Z is large or frequent occurring if its support is equal or greater than the minimum support threshold. An itemset of size k is called a k-itemset. The problem of mining association rules is divided into two subproblems[11]: (i) to find all frequent itemsets in the database for the given minimum support threshold value, and (ii) to generate the association rules using the frequent itemsets found in (i). As the mining association rules cost is mainly involved in(i), the focus is on the evolution of some efficient technique for the first subproblem [11].Distributed algorithm[1] for mining association rules statement:

To examine the association rules mining in a distributed databaseDB with D transactions and n-sites $S_1, S_2, \dots, S_n$ having n-partitioned$\{DB_1, DB_2, \dots, DB_n\}$respectively. Let D_i be the size of the partitions $DB_i$where i = 1, 2, . . . , n. Z.sup, the support counts of an itemset Zin database and $Z.Sup_i\ in\ DB_i$. For each site S_i, $Z.Sup_i$is the local support count of Z and Z.supis the global support count. For a specific minimum support threshold value‘s’, Zis also globally large itemset if $Z.sup \geq s \times D$; correspondingly, Zis locally large itemset at siteS_i, if $Z.sup_{\ i} \geq s \times D_i$. LetL be the globally large itemsets[1] in database, and $L_{(k)}$the globally large k-itemsets in L. A distributed association rule mining algorithm finds the globally large itemsets L.

3.2. POC Tree

Nodeset data structure[7] is based on the pre-order coding tree called POC-tree used in FIN algorithm. It is constructed with one root and prefix subtrees. Root is marked as null and item subtree as nodes. Database is scanned for frequent size - 1 itemsets and POC is constructed. It is used to generate nodeset of size-2 itemsets by preorder traversal. This is an efficient data structure, which use POC tree and reduces data scans and increase efficiency.

3.3.Candidate Set Pruning

Pruning is a process of reducing candidate sets[1] generated by data scan for itemsets size k=1,2,..n. It eliminates the frequent itemset which are not locally large itemset i.e. having support count less than the minimum support threshold as those may not be the global frequent itemsets. This reduces the number of candidates sets for communication to other nodes so reduces the communication load over the network and enhance the performance.

3.4. Load Balancing Technique

In the real life scenario database is distributed where data is captured or gathered at different locations. The size of the data partitions varies[19] in size from a few hundred of transactions on one site to a million of transactions at other site. In the distributed data mining resources are also distributed and there should be a mechanism to utilize all the nodes by allocating processing to less occupied nodes. In this paper a new technique for load balancing **Zero-first** for distributed data mining is presented with the following assumptions.

Assumptions:

- Database is partitioned and distributed at various sites around the globe
- Data is generated or captured at different sites and due to the resources constraints or policies, data may not be transferred to other sites
- Size of each partition may differ
- Size of Candidate set at different sites may differ
- All sites may not be equally loaded

Based on the assumptions Zero-first technique is developed for the assigning polling sites to each of the locally large candidate sets received from all the sites. The poling site is responsible for finding the globally large itemset from the list of locally large itemset. The new technique ensures the load is distributed to less occupied sites fora distributed data association rule mining.

Definition : Zero-first technique:

For sites$S = \{S_1, S_2, \dots, S_n\}$ and locally large candidate sets$\{CG_1, CG_2, \dots, CG_n\}$ received from n sites.

$$Candidate\ set\ CG = \{CG_1, CG_2, \dots, CG_n\}$$
$$max - CG = max\{CG_1, CG_2, \dots, CG_n\}$$

Where *max-CG* is the max number of candidates sets broadcasted by any site.

$$if\ max - CG > \ anyone\{CG_1, CG_2, \dots, CG_n\}$$
$$for\ \ C_i < max - CG \text{ Arrange}$$
$$S' = \ Order\{S_1, S_2, \dots, S_{\ n-q}\}$$
$$otherwise\ \ S' = \ Order\{S_1, S_2, \dots, S_{\ n}\}\ all\ sites$$

Sites in the order of size of the candidate sets starting with zero, excluding the sites with maximum size of candidate set. If candidate sets broadcasted by each site is not equal, then arrange all sites.

CG' = complete combined list of all locally large itemsets received from all the sites removing duplicates

$$CG' = \{CG_1 \cup CG_2 \cup \dots \cup CG_n\}$$
Allocate $CG'\ to\ sites\ S'$

After arranging the sites in order to their load, allocation of polling sites to the itemset are done in the order of the site occupancy (zero-first order) to check it for globally large itemset.[1]If all the frequent itemsets are not assigned then the repeat the assignment in the same sequence of initial assignment.

Let there are four sites S_1, S_2, S_3, S_4. The candidate sets broadcasted by site $S_1\{\ \ \}$, $S_2\{ab, bf\}$,$S_{\ 3}\{ab\}$, $S_4\{bc\}$. Applying zero-first technique, the ordered candidate set is $\{ab, bc,\ bf\}$ and ordered site set is $\{S_1, S_3, S_4\}$ leaving mist occupied site $\{S_2\}$. Using the zero-first technique, the first polling site is $\{S_1\}$and first locally large itemset in the ordered candidate set $\{ab\}$ is assigned to site $\{S_1\}$. Similarly $\{bc\}$ is assigned polling site is $\{S_3\}$and $\{bf\}$ itemset is assigned to $\{S_4\}$. Leaving the most occupied sites who broadcasted maximum number of locally large itemsets.

The sites with small data partitions have fewer number of transactions are under-utilized in terms of the processing, memory and scan time as compared to the sites with large data partitions. The Zero-first technique assigns more responsibility to less loaded sites and does not assigns any load to the most occupied sites. It ensures load balancing across the sites.

4. QDFIN : PROPOSED METHOD

In this section new algorithm QDFIN is proposed which uses efficient data structure nodeset at each site and construct the POC-tree[13]. It uses pruning technique locally and globally to reduce the candidate sets and zero-first technique for assigning polling site for load balancing. Globally large itemsets are computed at less loaded sites increasing the computational capacity.

Symbol Description[1]

s - Support threshold min-sup;
D - Number of transactions in database;
$L_k -$ Globally large k-itemsets;
Z.sup - Global support count of Z;
$CA_k -$ Candidate sets generated from L_k;
D_i - Number of transactions in DB_i;
$GL_{i(k)}$–globally large k-itemsets at S_i;
$CG_{i(k)}$ - Candidate sets produced by FIN algorithm;
$LL_{i(k)}$ - Locally large k-itemsets in $CG_{i(k)}$;
$Z.sup_i -$ Local support count of Z at S_i
$LP_{i(k)}$ – Local pruning k-itemset at site S_i

Algorithm -1: Zero-First Technique

$\boldsymbol{Input}$: $size - k\ locally\ large\ itemsets$
$and\ breadcasting\ site\ list \boldsymbol{S}_i (i = 1,2, \dots n)$

$\boldsymbol{Output}$: $polling\ site\ list\ for\ size$ –k itemset.

```
1.  $for\ all\ site$
2.     $if\ candidate\ set\ by\ sites\ are\ different\ size\ C_k$
3.         $arrange\ the\ sites\ in\ order\ of\ the\ number\ of$
                       $locally\ large\ itemset\ breadcasted\ Si$
4.         $remove\ the\ sites\ from\ list\ with\ max.number$
           $of\ locally\ large\ itemset\ breadcasted LL_{i(k)}$
5.  $for\ all\ \ locally\ large\ itemsets$
6.     $arrange\ in\ order\ removing\ duplicates\ LL_{i(k)}$
7.  $for\ all\ ordered\ candidates$
8.     $assign\ the\ polling\ site\ in\ zero - first\ order$
9.  $if\ candidate\ set\ exausited$
10.    $return\ polling\ site\ list\ for\ k - candidate\ set$
11. $else$
12.    $repeat\ the\ same\ order\ of\ sites\ and$
       $continue\ assignment$
```

Algorithm-2: QDFIN

$\boldsymbol{Input}$: $Partitioned\ \ database\ DB_i (i = 1,2, \dots n)$

$\boldsymbol{Output}$: L: $set\ of\ all\ globally\ large\ itemsets.$

$\boldsymbol{Method}$: $Execute\ the\ code\ for\ all\ k - itemsets$
$at\ all\ sites, starting\ from\ k = 1\ th\ size\ greater\ than\ 1.$

```
1)  $for\ all\ sites$
2)     $for\ k = 1$
3)         $find\ the\ support\ count\ T_{i(1)}$
4)         $construct\ the\ POC\ tree\ using\ FIN\ algorithm$
5)     $for\ k > 1$
6)         $find\ the\ size - 2\ itemset\ usin\ FIN\ algo$
7)         $(scan\ the\ POC\ to\ find\ size - 2\ itemset)$
8)  $for\ all\ itemsets\ Z\ belongs\ to\ frequent\ itemsets\ T_{i(k)}$
9)     $if\ support\ count\ is\ locally\ large\ then$
10)       $for\ all\ nodes$
11)            $insert\ itemset\ into\ locally\ large\ list\ \ LL_{i(k)}$
12)            $breadcast\ to\ all\ sites$
13) $for\ all\ sites$
14)    $for\ all\ itemsets\ Z\ belongs\ to\ locally\ large\ LL_{i(k)}$
15)       $insert\ itemset\ into\ local\ pruning\ set$
16) $using\ Zero - first, get\ the\ list\ of\ polling\ sites$
17) $for\ all\ sites,$
18)     $send\ locally\ large\ LL_{i(k)}\ to\ polling\ site\ S_m$
19) $for\ all\ itemsets\ Z\ belong\ to\ local\ pruning\ LP$
20)    $send\ polling\ request\ for\ itemset\ Z\ to\ all\ sites$
21) $all\ sites\ reply\ polling\ request\ from\ \ (T_{i(k)})$
22)       $send\ support\ counts\ Z.sup_m$
23) $for\ all\ itemsets\ Z\ in\ the\ polling\ set\ LP_{i(k)}$
24)    $receive\ support\ count\ Z.sup_m\ from\ all\ sites$
       $where\ itemset\ is\ not\ large$
25) $for\ all\ the\ itemsets$
26)       $calculate\ global\ support\ Z.\mathrm{sup}\ by$
          $sumup\ of\ all\ local\ support\ \ where$
27) $if\ Z.Sup > the\ global\ support\ threshold$
28)       $Add\ toglobal\ frequent\ itemset\ G_{i(k)}$
29)     $broadcast\ global\ frequent\ itemset\ G_{i(k)}$ ;
30) $if(k = 1)\ \ remove\_infrequent(DB_i)$;
31)       $Generate\ set\ of\ all\ globally\ large\ itemsets$
32) $return\ globally\ large\ k - itemset\ L_{(k)}$
```

The steps in the algorithm are explained below:

(i) Database DB_i for all partitions are scanned, local counts for all items of size-1 are found and POC-tree is constructed using FIN algorithm[7].This is responsible to generate candidate set $CG_{i(k)}$at all sites locally. If k>1 in the next pass, it uses POC-tree to find the candidate sets using FIN algorithm. If $CG_{i(k)}$ is empty, no k size itemsets are found then the process stops.(line 1-7)

(ii) At all the sites locally large itemsets of size-k are found by local pruning at all the sites where the count is more than the minimum local support threshold 's' and generate the locally large $LL_{i(k)}$itemset for broadcast.
The locally large itemset $LL_{i(k)}$are broadcasted to all other sites to get and inform the information about locally large for finding the globally large itemsets.(line 8-15)

(iii) Zero-first algorithm receives list of sites and locally large items communicating by each site. It returns the list of all polling sites for size-k locally large itemsets $LL_{i(k)}$which is communicated to all the sites $S_i$by the designated site.(line 16, call algorithm 1)

(iv) Polling sites store the itemsets in $LP_{i(k)}$ and store the list of sites and itemsets Z.large_sites. The polling sites$S_i$which receive these local frequent items $LL_{i(k)}$ send request to sites where these items were not frequent and gets the count of the items from remaining sites.Global counts of these items are found and the global frequent size-1 items are broadcasted to all sites.All these sites after receiving the request from the polling site check the structure and return the support count of these items to polling sites which are not locally large at that location.(line 21-25)

(v) At the polling sites after receiving all counts, computes the global counts for locally large items $LL_{i(k)}$. Then the global count of each candidate itemset is compared with the minimum support count condition to find the global large itemsets. The globally large itemsets are stored in $G_{i(k)}$ and broadcasted to all other sites. (line 25-29)

(vi) A home site receives the frequent itemsets. If it is the first pass then the dataset is updated. All the infrequent itemset of size-1 are removed from the database. A final set of global large itemsets are returned.After this to find the 2-itemset, all local large itemsets having size greater than 1, repeat the process step 1. Remove all infrequent items are removed having count less than minimum the globally large size-1 items. The POC-tree is scanned. This generates the 2-itemset and nodeset structure and so on. These local large itemsets are sent to respective polling sites.(line 30-32)

4.1. Efficiency of Local Frequent Mining

All the sites use efficient FIN[7] algorithm to find locally large itemsets and create an efficient POC-tree and nodeset data structure. All the frequent 1-itemsets are stored in the POC tree. For finding frequent 2-itemsets and nodeset the POC tree is scanned. Then delete the infrequent itemsets and initialize the nodesets of all frequent2-itemsets by null. Using the preorder traversal, generate the nodesets of all frequent 2-itemsets. Then the same procedure is used to generate the frequent-k itemsets. This reduces the number of database scans and improves the performance. The nodeset is an efficient data structure, this helps in further reducing the scan time. Table 1 is the part of the transaction database for size-**1** items. Figure1 shows the POC structure construction and nodeset for the data in Table 1.

Table 1: Database Transactions

TID	Items	Ordered frequent items
101	b, e, j, i, p	b, i
102	f, c, b, i	b, c, f, i
103	f, b, i, k	b, f, i
104	b, c, h,	b, c
105	F, a, b, c	a, b, c, f
106	A, f, c, g, b	a, b, c, f

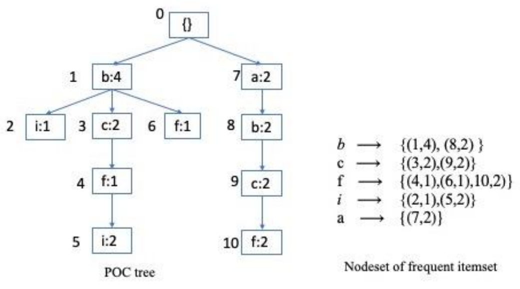

Figure 1: The POC-tree construction and Nodeset

4.2. Distributed Database and Resources

Data is distributed across the globe amongst different sites $\{S_1, S_2, \ldots, S_n\}$so called distributed database DB_i. Each site finds the local frequent itemset at each site LL_i. Each site is being used and gives throughput proportional to the number of nodes or sites. There is a tradeoff between the communication load for processing throughput in order to get the best performance out of the setup depending of number of sites. There is not only one centralized site which processes and finds the frequent itemsets rather all sites behave as home as well as poling site. Sites having small data portion a few transactions. It takes less time to scan and create nodeset, also less capabilities and memory to process the small data as compared to sites with large data. These nodes are under-utilized. Zero-first technique takes care of the highly loaded sites and assign polling to less loaded sites first for generating the globally large itemsets. It is the best load balancing technique which utilizes the less loaded sites by effective use of resources. All the sites are involved in processing the support count and finding the global frequent itemsets for a specific local large itemset hereby reducing the load of finding global frequent itemset on one centralized site. In this algorithm all sites participate in the process of the local and them global frequent itemsets and good amount of parallelism is achieved.

4.3. Communication Load Reduction

At each site there is a process of pruning which reduces the size of the candidate sets. Whenever any site finds the candidate sets of frequent itemsets, if it is not locally large to that particular site, that site remove that itemset from the candidate set by the means of local pruning. If frequent size-2 itemset at site -2 {ab, ac, bf, cf}. After pruning itemsets having support count less than the minimum support threshold say {cf, bf}are removed from the candidate sets. The remaining candidate sets {ab, bf} communicated to all other sites.The pruning process reduces the number of candidates sets drastically hence reduce the load on the network and communication cost. This process makes this network efficient algorithm as small set of data is being communicated to all sites.

5. EXPERIMENTAL EVALUATION

5.1.Experimental Setup

This section deals with the environment used to run the algorithm and it is then evaluated depending upon various parameters and compared with FDM-FP and PFIN[24] on datasets two datasets. Experiments are performed on Eclipse Indigo with clusters of nodes varying four, five and six nodes with windows 10 OS, with RAM 6 GB each and hard-disk 1 TB system 64 bit running at 3.30 GHz having Java JDK 1.7

5.2. Datasets

The algorithms are run on Mushroom datasets. These datasets are downloaded from FIMI data repository[30]. Mushroom dataset is build using characteristics of mushrooms and dataset specifications are:

- Average length = 23
- Number of Items = 119

- Total Transactions = 8124

Two experiments are performed, one on equal data size partitions at each site and another on varying size data partitions in the following sizes.

Experiment 1: Uniform data partitions size shown in Table 2

Table2: Uniform Data Partition size at different site

No. of Nodes	DB_1	DB_2	DB_3	DB_4	DB_5	DB_6
4	2030	2030	2030	2034		
5	1625	1625	1625	1625	1624	
6	1350	1350	1350	1350	1350	1374

Experiment-2: Varying data partition size shown in Table 3.

Table3:VaryingData Partition size at different sites

No. of Nodes	DB_1	DB_2	DB_3	DB_4	DB_5	DB_6
4	500	1500	2500	3624		
5	300	900	1500	2100	3324	
6	100	600	1100	1600	2100	2624

5.3. Performance Analysis

The proposed algorithm QDFIN is compared with FDM-FP (FDM using FP Growth) and PFIN(Parallel FIN)[24] algorithms on the basis of execution time.In the proposed algorithm all sites participate in the processing giving better throughput and pruning technique reduces the number of candidate sets resulting low communication overhead. It assigns the polling site using new presented technique zero-first which further reduces load on the fully occupied sites. At each site an efficient data structure Nodeset based on POC-tree[7]is used which reduces number of data scans and hence improve the performance by reducing the data access time.

The algorithms are run on Mushroom dataset with minimum supports thresholds of 10, 20, 30, 40, 50, and 60 percent. Both experiments are done on 4, 5, and 6 nodes setups and are compared on the basis of execution time.

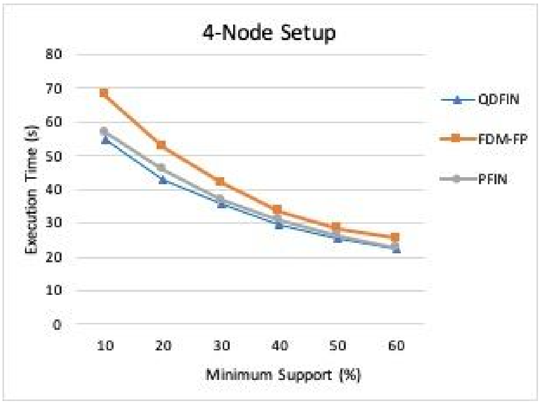

Figure 2: Execution time on uniform partition size on 4 nodes

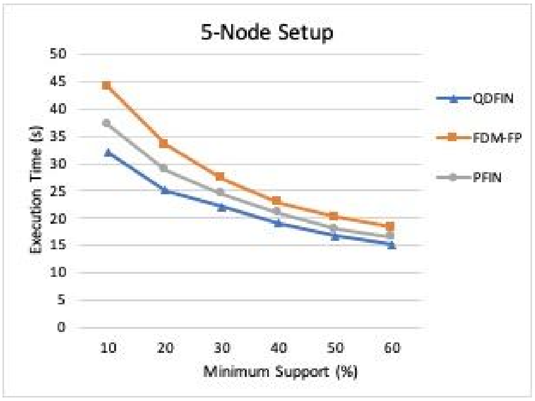

Figure 3: Execution time on uniform partition size on 5 nodes

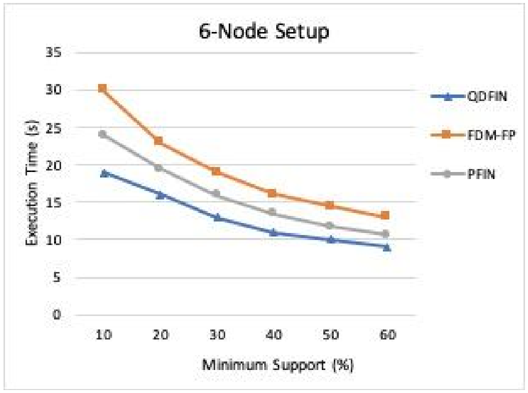

Figure 4: Execution time on uniform partition size on 6 nodes

Execution on Uniform data partition size

In the first experiment where each site is having uniform partitions size, QDFIN performs better in all 4, 5, and 6node setups shown in Figures 2, 3, and 4. Figure 2 shows in 4-node setup, execution time of QDFIN is close to the execution time of PFIN algorithm as both the algorithm use nodesets data structure for finding locally large itemsets, which reduces the scan time. But it performs better than the FDM-FP where FP-tree structure is used. In the 4-node setup all the sites are producing candidate sets and the advantage of the zero-first technique is very small. The same algorithms are also compared in the 5-node and 6-node setups.Figures3 and 4show that the proposed algorithm QDFIN performs best in 5 nodes. It further improves in 6-node setup as with the increase of number of nodes, number of locally large k-itemsets further reduces with the increase of k. It makes some imbalance in the sites in broadcasting the number of candidate sets where some sites become less occupied or free. The zero-first technique assigns the polling site in order of their occupancy for finding globally large itemsets. Figures 3 and 4 shows the same effect where the performance of the QDFIN is better than the other algorithms.

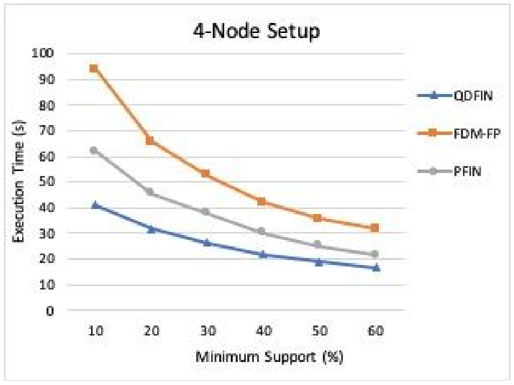

Figure 5: Execution time on varying partition size on 4 nodes

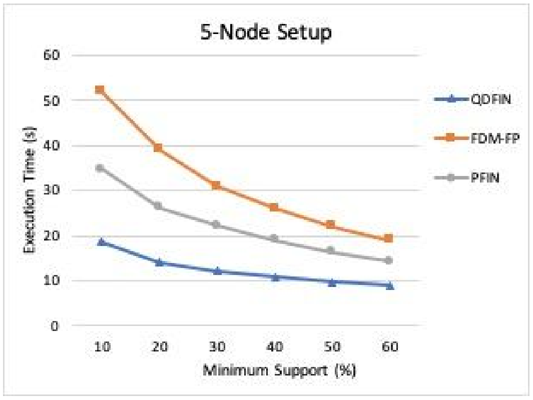

Figure 6: Execution time on varying partition size on 5 nodes

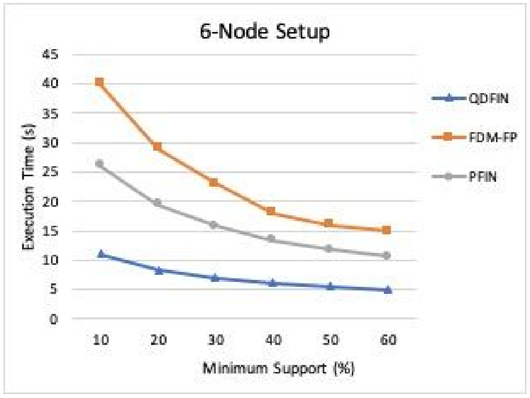

Figure 7: Execution time on varying partition size on 6 nodes

Execution on varying data partition size

In the second experiment, same algorithms are compared with the varying size of the data partitions on various sites. Figures 5,6, and 7 show that the performance of the QDFIN is best in all three setups. The time performance of QDFIN is much better due to the reason of load balancing. As the data size available at all nodes differ implies that number of transactions differ. The sites with fewer number of transactions produces small candidate sets, takes less time to scan the data, use lesser memory, less processing required. QDFIN takes the advantage of the load difference and assigns less loaded sites first as polling site. Some sites have large number of transactions takes more time and processing capabilities to scan, store the data and generate more candidate sets and are highly busy. The zero-first technique of the proposed algorithm excludes these busy sites from the polling site list. Number of sites increases in Figures 6, 7 and the difference in number of candidate sets generated by different sites also increase. Some of the sites generate no candidate sets even for k=1 or k=2, this imbalance further increase in case of 6-node setup as shown in Figure7. It directly effects the load balance, which makes QDFIN best amongst the all three algorithms. The performance difference as compared to FDM-FP is even more due to the efficient data structure nodesets being used in QDFIN as compared to FP-Tree. It is also observed that with low minimum support threshold say 10%, 20%, high number of frequent itemsets are generated and all algorithms take more time and QDFIN performs better due to the difference of candidate set generations amongst each site. The time performance difference reduces for higher minimum support threshold50%, 60% because smaller number of candidate sets are generated, even some sites generate zero frequent itemsets and QDFIN performs better by load balancing.

QDFIN performs best in lower as well as higher minimum support count in both the experiments i.e. with varying or uniform size data partitions in all three setup 4, 5, and 6 nodes as compared to the PFIN and FDM-FP algorithms. It also shows that with the increase of the number of nodes or decrease in the minimum support threshold QDFIN outperforms other similar algorithms specially in varying data size. It uses the advantages of the POC-tree and nodeset data structure locally by saving scan time, pruning in reducing communication and zero-first technique for load balancing.

6. CONCLUSIONS

In this paper, new algorithm QDFIN for distributed association rule mining on distributed data of varying partition size is proposed. The proposed algorithm uses the efficient data structure, nodeset[7] to generate the candidate itemsets at each site locally, the pruning of data for low communication load and zero-first technique for load balancing. The algorithm performance is evaluated on three different setups of 4-nodes, 5-nodes, and 6-node with varying support threshold. Two experiments are performed one with uniform and one with varying data partition size. The performance of the proposed algorithm is compared with some of the similar algorithms FDM-FP and PFIN[24] algorithms.

The QDFIN outperforms the existing algorithms in the execution time comparisons in all setup, especially in varying data partition size in all 4, 5, and 6 node setups. The new zero-first technique is very useful in the real life scenario where the data is skewed i.e. not uniformly distributed amongst the sites and even not possible to balance the same. It performs better with the increase of number of sites or nodes as it overcome the disadvantage of the data skew.

In future this algorithm can be used in larger setup and also for large scale distributed association rule mining problems and for large datasets. The resources and the capabilities of the nodes can be considered for further improvement.

REFERENCES

1. D. W. Cheung, Jiawei Han, V.T. Ng, A. W. Fu, and Yongjian Fu, "**A fast distributed algorithm for mining association rules**", *Fourth International Conference on Parallel and Distributed Information Systems*, IEEE, 1996.
2. Agrawal R. and Ramakrishnan Srikant, "**Fast algorithms for mining association rules**", in *Proc. International Conference on Very Large Scale Data Base*, 1994, pp. 487-499.
3. Ashrafi, Mafruz Zaman, David Taniar, and Kate Smith, "**ODAM: An optimized distributed association rule mining algorithm**", *IEEE distributed systems online*, 2004. https://doi.org/10.1109/MDSO.2004.1285877
4. P. Naresh and Dr. R. Suguna, "**Association rule mining algorithms on large and small datasets: a comparative study**", in *Proc. International Conference on Intelligent Computing and Control Systems (ICICCS 2019) IEEE*: *CFP19K34-ART*, pp. 587-592.
5. Han, Jiawei, Jian Pei, and Yiwen Yin., "**Mining frequent patterns without candidate generation**", in *Proc. of the 2000 ACM SIGMOD international conference on Management of data*, vol. 29. no. 2, pp. 1-12, 2000. https://doi.org/10.1145/335191.335372
6. Deng, Zhi-Hong and Sheng-Long Lv., "**PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning**", *Expert Systems with Applications*, vol.42 no.13, pp. 5424-5432, 2015.
7. Deng, Zhi-Hong and Sheng-Long Lv., "**Fast mining frequent itemsets using Nodesets**", *Expert Systems with Applications*, vol. 41 no. 10, pp. 4505-4512, 2014.
8. F. Bodon and L. Rónyai, "**Trie: An alternative data structure for data mining algorithm**", *Mathematical and Computer Modelling*, vol. 38, Issue 7–9, pp. 739-751, 2003.
9. Vinaya Sawant and Ketan Shah, "**A survey of distributed association rule mining algorithms**", *Journal of Emerging Trends in Computing and Information Sciences*, vol. 5, pp. 391-398, 2014.
10. Han, Jiawei, et al., "**Frequent pattern mining: current status and future directions**", *Data Mining and Knowledge Discovery*, vol.15.1, pp. 55-86, 2007. https://doi.org/10.1007/s10618-006-0059-1
11. Agrawal R., Tomasz Imieliński, and Arun Swami, "**Mining association rules between sets of items in large databases**", *ACM SIGMOD Record* 22.2,pp. 207-216, 1993.
12. Ailing Wang, "**An improved distributed mining algorithm of association rules**", *Journal of Convergence Information Technology,* vol. 6, no.4,pp.118-122, 2011.
13. Deng, ZhiHong, ZhongHui Wang, and JiaJian Jiang, "**A new algorithm for fast mining frequent itemsets using N-lists**", *Science China Information Sciences*, vol.55, no.9, pp. 2008-2030, 2012.
14. Deng, Zhi-Hong, "**DiffNodesets: An efficient structure for fast mining frequent itemsets**", *Applied Soft Computing*, vol. 41, pp. 214-223, 2016. https://doi.org/10.1016/j.asoc.2016.01.010
15. Pyun, Gwangbum, Unil Yun, and Keun Ho Ryu, "**Efficient frequent pattern mining based on linear prefix tree**", *Knowledge-Based Systems*, vol. 55, pp. 125-139, 2014.
16. Van Quoc Phuong Huynh, Josef Küng, Markus Jäger and Tran Khanh Dang, "**IFIN$^+$a parallel incremental frequent itemsets mining in shared- memory environment**",in *Proc. International Conference on Future Data and Security Engineering FDSE 2017*,pp. 121-138.
17. Van Quoc Phuong Huynh, Josef Küng and Tran Khanh Dang, "**A Parallel incremental frequent itemsets mining IFIN$^+$: improvement and extensive evaluation**",*Transactions on Large-Scale Data-and Knowledge-Centered Systems XLI*,pp. 78-106, 2019.
18. Vinaya Sawant and Ketan Shah, "**Performance evaluation of distributed association rule mining algorithms**", *7th International Conference on Communication, Computing and Virtualization, Procedia Computer Science* 79, pp. 127-134, 2016. https://doi.org/10.1016/j.procs.2016.03.017
19. Asma Belhadi, YoucefDjenouri, Jerry Chun-Wei Linand Alberto Cano, "**A general-purpose distributed pattern mining system**", *Springer-Applied Intelligence*, vol. 50, pp. 2647–2662, 2020.
20. George Gatuha and Tao Jiang, "**Smart frequent itemsets mining algorithm based on FP-tree and DIFFset data structures**", *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 25, pp. 2096-2107, 2017.
21. Norulhidayah Isa, Nur SyuhadaMohd Yusof and Muhammad Atif Ramlan, "**The implementation of data mining techniques for sales analysis using daily sales data**", *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no.1.5, pp. 74-80, 2019. https://doi.org/10.30534/ijatcse/2019/1681.52019
22. Agrawal R. and Shafer John C., "**Parallel mining of association rules**", *IEEE Transactions on Knowledge & Data Engineering*, vol. 8, Issue 6, pp. 962-969, 1996.
23. David Cheung, Vincent T.Y. Ng, Ada W. Fu and Yongjian Fu, "**Efficient mining of association rules in distributed databases**", *IEEE Transactions on Knowledge and Data Engineering,* vol. 8 no. 6, pp. 911-922, 1996.
24. Chen Lin and Junzhong Gu, "**PFIN: A parallel frequent itemset mining algorithm using nodesets**", *International Journal of Database Theory and Application*, vol. 9, no.6, pp. 81-92, 2016. https://doi.org/10.14257/ijdta.2016.9.6.08
25. Liao, Jinggui, Yuelong Zhao and Saiqin Long., "**MRPrePost—A parallel algorithm adapted for mining big data**", *IEEE Workshop on Electronics, Computer and Applications(IWECA)*, 2014.
26. Thakare, Sanket, Sheetal Rathi, and R. R. Sedamkar, "**An improved Prepost algorithm for frequent pattern mining with Hadoop on cloud**", *Procedia Computer Science*, vol. 79, pp. 207-214, 2016.
27. Nader Aryabarzan, Behrouz Minaei-Bidgoli and Mohammad Teshnehlab, "**negFIN: An efficient**

algorithm for fast mining frequent itemsets",*Expert System and Applications*, vol. 105, pp. 129-143, 2018.
28. NVS Pavan Kumar, Dr. J K R Sastry and Dr. K Raja Sekhara Rao, "**Mining distributed databases for negative associations from regular and frequent patterns**", *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, no. 4, pp. 1449-1463, 2019. https://doi.org/10.30534/ijatcse/2019/64842019
29.Jun Liu, Yuan Tian, Yu Zhou, Yang Xiao and Nirwan Ansari, "**Privacy preserving distributed data mining based on secure multi-party computation**", *Computer Communications*, vol.153, pp.208-216, 2020. https://doi.org/10.1016/j.comcom.2020.02.014
30. **FimiDataset Repository**: Online Portal http://fimi.ua.ac.be/data/