A New Dynamic Distributed Algorithm for Frequent Itemsets Mining

Azam Adelpoor
Department of Computer, Science and Research BranchIslamic Azad University, Khouzestan, Iran

Mohammad Saniee Abadeh
Department of Electrical and ComputerEngineering TarbiatModares University, Tehran, Iran

ABSTRACT
Mining for association rules between items in large transactional databases is a central problem in the field of knowledge discovery. It has crucial applications in decision support and marketing strategy. Centralized and Distributed Association Rules Mining (DARM) include two phases of frequent itemset extraction and strong rule generation. The most important part of ARM is Frequent Itemsets Mining (FIM)and because of its importance in recent years, there have been many algorithms implemented for it. In this paper, we have focused on distributed Apriori-Like frequent itemsets mining and proposed a distributed algorithm, called New Dynamic Distributed Frequent Itemsets Mining (NDD-FIM), for geographically distributed data sets. NDD-FIM has a merger site to reduce communication overhead and eliminates size of dataset partitions dynamically. The experimental results show that our algorithm generates support counts of candidate itemsets quickerthan other DARM algorithms and reduces the size of average transactions, datasets, and messageexchanges.

Keywords
Distributed Data Mining;Frequent Itemsets;Association Rule; Apriori Algorithm

1. INTRODUCTION
Frequent itemsets mining is at the core of various applications in the data mining area. It is majorly applied in association rules mining [1,2], correlation analysis, sequential patterns mining [3], multi-dimensional patterns mining [4], among others.

The best known such task is the association rules discovery. Anassociation rule is a rule which implies certain association relationships among a set of items in a database. The meaning of an association $X \Rightarrow Y$, where X and Y are set of items, is that transactions of the database which contain X tend to contain Y. However, there could be a lot of associations among the data which may not be able to deduce from common knowledge. Therefore, since association rules′s inception, many sequential and distributed frequent itemset mining algorithms have been proposed in the literature [5-15], etc. Many of them are correlated to the Apriori algorithm [10] which is a well-known method. However, it is costly to find candidate itemsets. Thus, many researchers have been trying to improve it.

Basically, frequent itemsets generation algorithms search the dataset to determine which combination of items occurs together frequently. Considering the commonly known market basket analysis; ARM analyses customer buying habits by finding frequent itemsets and associations between the different items that customers place in their "shopping baskets". For instance, if customers are buying milk, how likely are they going to also buy bread on the same trip to the supermarket? Each customer buys a set of items as his/her basket that is a transaction. For a fixed threshold support s, the algorithm determineswhich sets of items, of a given size k, are contained in at least s of the $t$transactionsor baskets.

Such information can lead to increased sales by helping retailers do selective marketing and arrange their shelf space.

Most enterprises collect huge amounts of business data from daily transactions and store them in distributed datasets; specially, for security issues and communication overhead, those distributed datasets are usually not allowed to be transmitted or joined together, therefore, in this study we are focusing on Apriori-based algorithms and discovering frequent itemsets on extremely large and distributeddatasets over different geographic locations and will present a well-adapted distributed approach for this purpose, based on both analytical and experimental approaches.

Like other data mining techniques that must process enormous datasets, FIM is inherently disk-I/O intensive.These I/O costs can be reduced byeliminating infrequent items, finding more identical transactions, and reducing the number of times the database needs to be scanned.

In a distributed environment the practical implications of communication overhead, the effect of the underlying architecture, and the dynamic behavior of the system are issues that contribute to the complexity of a distributed environment [16].

The partitioning of a task and subsequent migration of some of the resulting subtasks will result in increased communications overhead. Therefore, a most important problem of distributed architectures and algorithms is overhead′s cost in inter-site communications among processors. Some of the distributed algorithms like CD [5], FDM [6], and FPM [8] do not have any merger site and message exchange is all to all which is not a strong architecture. But, DDM [9] and ODAM [15] have used a merger site to collect local counts from other sites. This method reduces communication overhead significantly when compared to other algorithms. Generally speaking, the algorithm must balance the overhead communication cost against the resulting improvement in the throughput.

The worst-case work and communication needed by a classical distribution of the Apriori algorithm can be exponential in the size of the input. Considering the case where every transaction, in every node, contains every item, the algorithm must output and may communicate each subset of I items, at each level. This can be due to remote support counts collections for global pruning purposes. In this case, many of the communications are completely unnecessary.

Based on these observations, the proposed approach has three phases: the local mining phase, the communication phase, and the global mining phase. Also elimination of infrequent items and finding similar transaction is a dynamic background action. In the first phase, we consider only counting and a local pruning strategy. In thesecond phase, each node sends support counts of a collection of locally frequentitemsetsto the merger node. The merger node collects frequent counts and asks other node by necessity. The overhead related to communicationphase in

classical approaches can then be highly reduced using a constrained collectionphase with much fewer passes. Moreover speed of the counting phase will be increased by elimination of the infrequent items.
Also generation of candidate itemsets will be performed by the merger site not by all nodes.
While our performance study focuses on the Apriori-baseddistribution, we believe that the key reasoning of this study will hold for many other frequentitemsets generation tasks, since it is partly related to the dataset properties.
The rest of this article is organized as follows. Section 2 provides the preliminaries of basic concepts and their notations to facilitate the description the well-known algorithms. Section 3 surveys works related to distributed frequent itemsets mining. In Section 4, we define our proposed algorithm in detail. Section 5 reports the experimental results. Finally the conclusion of this work and future works are given in Section 6.

2. PROBLEM DEFINITION

In this Section, we define frequent itemsets mining problem, its distribution aspect, and properties.
The frequent itemsets generation problem can be stated as follows.Let $I=\{i_1,i_2,\ldots,i_m\}$ be a set of *m*items and *D* be a database of transactions, where each transaction *T* consists of a set of items such that $T \subseteq I$. Given an itemset$x \subseteq I$ of size *k* that is known as *k*-itemset, a transaction *T* contains *x* if and only if $x \subseteq T$. For an itemset*x*, the support of *x*denoted as $s_x(D)$, is defined as the number of transactions in *D*which *x* occurs as a subset. Let *minsup* be the minimum support threshold specified by user. If $s_x(D) \geq minsup$, *x* is called a frequent itemset [17].
A frequent itemset is maximalif it has nosuperset that isfrequent. Some approaches are dedicated to this problem [18,19,20].
A typical architecture of a distributed data mining approach is depicted in Figure 1.The first phase involves the analysis of the local database at distributed sites. Then, the discovered knowledge is usually transmitted to a central site, and the integration is performed. The results are transmitted back to the local databases.
In some approaches [5,6,8], instead of a merger site, the localmodels are broadcasted to all other sites, so that eachsite can in parallel compute the global model.

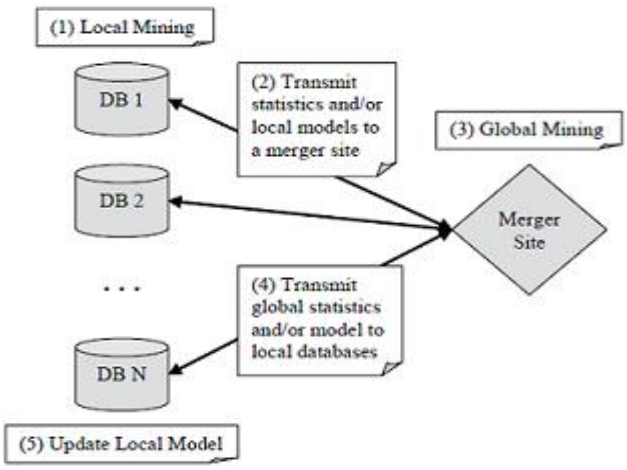


Fig 1: Typical architecture of Distributed Data Mining approaches [21]

The distribution aspect of FIM can be described as follows. Let *D*be a dataset of transactions partitioned horizontally over *M* nodes $\{n_1,n_2,\ldots,n_m\}$, and the size of the partition $n_i$be D_i. Let $s_x(D)$ and $s_x(D_i)$ be the support count of the itemsetxin *D*andD_i, respectively. For a given minimum support thresholds_d, called *minsup*, an itemsetxis *globally frequent* if $s_x(D)$ is greater than $s_d \times |D|$, and is*locally frequent* at a node n_i if$s_x(D_i)$ is greater than $s_d \times |D_i|$.
Two basic properties are described here:

Property 2.1 A globally frequent itemset must be locally frequent in at least one node.
Proof Let *x* be an itemset. If $s_x(D_i)$ is smaller than $s_d \times |D_i|$ for i = 1,…,M, then $s_x(D)$ is smaller than $s_d \times |D|$ (since $s_x(D)=\sum_{i=1}^{M} s_x(D_i)$ and $D=\sum_{i=1}^{M} D_i$), and *x* cannot beglobally frequent. Then, if *x* is a globally frequent, it must be locally frequent in at least onenoden_i.

Property 2.2 All subsets of a globally frequent itemset are globally frequent.
Proof Let *x* be an itemset, and *x*′be a subset of *x*. If $s_{x'}(D)$ is smaller than $s_d \times |D|$,then $s_x(D)$ is also smaller than $s_d \times |D|$(since $s_x(D) \leq s_{x'}(D)$), and *x* cannot be globallyfrequent. Then, if *x* is globally frequent, all its subsets must be frequent.

3. PREVIOUS WORKS

Algorithms for the distributed FIM problem usually are parallelization of sequential FIM algorithms. CD, FDM, FPM and DDM [5,6,8,9] parallelize Apriori [10], PDM [12] parallelizes DHP [11], D-Sampling [7] is a combination of serialsampling approach [22] and DDM algorithm, parallel FP-growth [13] is a parallelized version of FP-growth [14], and so on.As mentioned before, many frequent itemsets mining algorithms, both sequential and distributed, are related to the Apriori algorithm [10]. The name of the algorithm is based on the fact that it uses prior knowledge of frequent itemsets properties. It exploits the observation that all subsets of a frequent itemset must be frequent.Apriori is a serial algorithm that has a smaller computational complexity when compared with other serial algorithms [23]. This algorithm performsthreesteps in Fig. 2. ($L_k$andC_k are frequent and candidate itemsets of size k, respectively).Step 2 is named as *apriori_gen* function which is used in the other Apriori-Based algorithms and is stated in Fig. 3.

1. GenerateL_1, then iterates steps 2, 3 and 4
2. Generate$C_k$from L_{k-1}
3. Examine $C_k$to determine whether the candidate sets meet the *minsup*or not. If the answer is positive, add itemsetto L_k
4. Stop if $L_k = \varnothing$

Fig2:Apriorialgorithm [10]

Procedure apriori_gen(L_{k-1},*min_sup*)
1.for each itemset$l_1 \subseteq L_{k-1}$
2. for each itemset$l_2 \subseteq L_{k-1}$
3. If ($l_1[1] = l_2[1]$)^($l_1[2] = l_2[2]$)^…^
($l_1[k\text{-}2]=l_2[k\text{-}2]$)^($l_1[k\text{-}1] \neq l_2[k\text{-}1]$) then {
4. $c= l_1 \infty l_2$; // join step
5. Ifhas_infrequent_subset(c,L_{k-1})
6. Then delete *c*; // prune step
7. Else add *c* toC_k;
8. }
9.ReturnC_k

Fig3:apriori_genfunction [10]

The CD (Count Distribution) and DD(Data Distribution) algorithms [5]are simple parallelization of the Apriorialgorithm, and assume data sets are horizontally partitioned amongdifferent nodes and each node has a copy of candidate itemsets.CD doesn't exchange data tuples betweenprocessors, and only exchanges the counts. Each processor only needs to process the data it owns, and generates its local candidateitemsets depending on its local partition. Each node obtains global counts byexchanging local counts with all other processors. The CD's communication complexity is $O(|C_k|n^2)$inpass *k*, where $|C_k|$ and *n* are the size of candidate *k*-itemsets and number of local sites,respectively.The amount of communication, however, increases with processors increased.The program fragment of CD at processor p_i for the *k*th iteration is outlined in Fig. 4.

---

1. GenerateC_k from L_{k-1}

2. Scan$D_i$to find $s_x(D_i)$ for all $x \in C_k$

3. Exchange $s_x(D_i)$ with all other sites to calculate $s_x(D)$

4. $L_k = \{x \in C_k \mid s_x(D) \geq s_d \times |D|\}$

5. Stop if $L_k = \varnothing$

Fig4:CD algorithm [5]

The other one, DD, partitions the candidate itemsets among the processors and improves the memory usage rather than CD.It is needs to scan the rest of the transactions stored in the memory of the other processors in addition to the locally assigned transactions. This algorithm was found to be slower than the CD, because of each processor sends to all the other processors the portion of the database that resides locally and this manner has a high communication overhead.

In order to reduce the communication overhead, FDM was proposed in [6]. It is based on the fact that a globally frequent itemset must be locally frequent in at least one node. Thus, in FDM, every node finds locally frequent itemsetsin its partition and exchanges to other nodes. Next, support counts are globally summed forthose candidate itemsets which are locally frequent by at leastone site. Global frequent itemsetsare used to generate the next level candidates.

If the probability that an itemset has the potential tobe frequent is $Pr_{potential}$ then the communication complexity of FDM is $O(Pr_{potential}|C_k|n^2)$ in pass *k*. A comparison of CD and FDM based on candidate set, message size reduction, and execution time reduction, shows FDM as performing better. The main problem with FDM is that $Pr_{potential}$ is not scalable in *n* and it quickly increases to 1 as *n* increases [8].

Another algorithm that is based on Apriori is Distributed Mining of Association rules (DMA) algorithm [29]. It is similar to FDM but uses polling sites to optimize the exchange of support counts among sites and reducing the communication complexity in pass *k* to $O(|C_k|n)$.

FDM was further enhanced into another efficient parallel algorithm; FPM (Fast Parallel Mining) [8]. It hasincorporated two pruning techniques, distributed pruning and global pruning, and generates candidate itemsetsless than FDM.For the *k*th iteration of FPM, $k > 1$, the program fragment executed at $p_i$is described in Fig. 5 ($GL_{k(i)}$ is used todenote gl-large *k*-itemsets at processorp_i. If an itemset*x*is both globally large and locally large at processor p_i, *x* is called gl-large at processorp_i).

Another Apriori-based algorithm, the Optimized Distributed Association rules Mining (ODAM), is proposed in [15]. It follows the paradigm of FDM and eliminates all infrequent items after the first pass to reduce average size of transactions and database and efficiently generate candidate support counts in latter passes. Nevertheless, when ODAM removes infrequent 1-itemsets from database, the chance of finding similar transactions increases. Also, at the communication level, it minimizes the total message exchange by sending support counts of candidate itemsets to a single site, called receiver. The receiver broadcasts the globally frequent itemsets back to the distributed sites.

---

1. $C_k = \cup_{i=1}^{M} apriori\_gen(GL_{k-1(i)})$

2.Prune $C_k$by global pruning

3.ScanD_i to find $s_x(D_i)$ for all $x \in C_k$

4.Exchange $s_x(D_i)$ with all other sites to calculate $s_x(D)$

5. $GL_{k(i)} = \{x \in CG_k \mid s_x(D) \geq s_d \times |D|,\ s_x(D_i) \geq s_d \times |D_i|\}$ for all *i=1,...,M*

6. $L_k = \cup_{i=1}^{M} GL_{k(i)}$

7. Stop if $L_k = \varnothing$

Fig5:FPM algorithm [8]

Another parallel-distributed algorithm is proposed in [24] based on the Trie tree [25]. This algorithm distributes workloads according to the first level of the Trietree to balance and speed-up the computation. However, this may cause the sizes of candidate itemsets (workloads) among processors significantly varying.

An efficient frequent-pattern mining algorithm, called EDMA [26], is proposed based on the Apriori. EDMA uses the CMatrix data structure to store the transactions and minimizes the number of candidate sets and reduces the exchange messages by local and global pruning. The execution time gets longer when the database size is larger, since EDMA will access CMatrix a lot of times when calculating candidate itemsets.

In [27] a Dynamic Distributed Rule Mining (DDRM) is implemented that is a dynamic extension of Prefix-based [31] algorithm and has used a lattice-theoretic approach for mining association rules. Actually DDRMpartitions the lattice into sub lattices to be assigned to processors for processing and identification of frequent itemsets. At first phase, the partitions are transformed from horizontal format to a vertical Tid-list format, and the candidates are counted by intersecting the Tid-lists. Unlike high performance, transferring Tid-list between local nodes and the controller node has huge amounts of overhead. Another problem is thatevery nodescans full Tid-List which is collection of local Tid-List, to process assigned sub lattice.

4. THE NDD-FIMALGORITHM

A distributed FIM algorithm will performs better if we can reduce communication cost, idle time, wait time, and number of dataset scans.The performance of Apriori-based algorithms degrades for said various reasons.We need to focus on these problems.

4.1 Reduction of number of dataset scans

All Apriori-based algorithms require*k* number of database scans to generate a frequent *k*-itemset. To overcome this problem, ODAM algorithm [15] eliminates all infrequent 1-itemsets after the first pass and generates candidate support counts of later passes efficiently. The number of items in the dataset might be large, but only a few will satisfy the support threshold [28].

This technique not only reduces the average transaction length but also reduces the dataset size significantly. Nevertheless, by elimination of infrequent 1-itemsets, the chance of finding similar transactions increases.

We have extended this issue to all passes and eliminated all items which have not participated in production of frequent itemsets. The method of calculation of non-frequent items in the first pass (NL_1) and non-frequent items in the *k*th pass (NL_k) has shown as follows:

$NL_1 = \{i_i | (i_i \square I) \wedge (i_i \notin L_1)\}$

$NL_k = \{i_i | (i_i \square I) \wedge (i_i \notin NL_{k-1}) \wedge (\nexists l \square\square L_k \square\square\square i_i \subset l)\}\ k \neq 1$

Consider the sample dataset in Table 1 and specified minimum support 0.5. After first pass I_5 and I_6 are infrequent 1-itemset. We eliminate them and find more identical transactions (Table 2).

Table 1.Sample dataset

Tr. No.	Tr. count	Items
1	1	$I_1I_2I_3$
2	1	$I_2I_3I_6$
3	1	I_1I_2
4	1	$I_1I_2I_3I_5$
5	1	$I_1I_3I_4$
6	1	$I_1I_2I_6$
7	1	$I_3I_4I_5$
8	1	I_1I_4
9	1	$I_2I_3I_5$
10	1	$I_1I_2I_3I_4$

Table 2.Sample dataset after elimination of I_5 and I_6in first pass

Tr. No.	Tr. count	Items
1,4	2	$I_1I_2I_3$
2,9	2	I_2I_3
3,6	2	I_1I_2
5	1	$I_1I_3I_4$
7	1	I_3I_4
8	1	I_1I_4
10	1	$I_1I_2I_3I_4$

ODAM stops elimination here, but we keep this technique in all of the next passes. At second iteration, we have 2-itemsets $\{I_1I_2,\ I_1I_3,\ I_1I_4,\ I_2I_3,\ I_2I_4,\ I_3I_4\}$that only I_1I_2 and I_2I_3 is frequent. Furthermore, $NL_2 = \{I_4\}$ and we can eliminate I_4 from dataset (Table 3). This technique not only reduces average transaction size but also finds more identical transactions.
Also, at *k*th iteration, transactions that are smaller than or equal to *k* can be eliminated. This manner is effective on final iterations of mining and real world datasets which variation of transactions′s size is tremendous. Table 4 shows the result of elimination of short transactions in Table 3.

Table 3.Sample dataset after elimination of I_4 in second pass

Tr. No.	Tr. count	Items
1,4,10	3	$I_1I_2I_3$
2,9	2	I_2I_3
3,6	2	I_1I_2
5	1	I_1I_3
7	1	I_3
8	1	I_1

Table 4.Sample dataset after reduction of transactions smaller than or equal to 2

Tr. No.	Tr. count	Items
1,4,10	3	$I_1I_2I_3$

4.2 Reduction of Communication Cost

A typical architecture of a distributed data mining approach wasshown in Figure 1. In summeryas mentioned before, in some approaches [9,15,27], the analysis of the local database at distributed sites is transmitted to a central site, and the integration is performed.For FIM problem, in this situation, the communication complexity is $O(|C_k|n)$inpass *k*, where $|C_k|$ and *n* are the size of candidate *k*-itemsets and number of local sites, respectively.
In some other approaches [5,6,8], instead of a merger site, the local models are broadcasted to all other sites, so that each site can in parallel compute the global model.The communication complexity is $O(|C_k|n^2)$. Futhermore, we have used a merger site in NDD-FIM to reduce communication cost.
On the other hand, using a powerful pruning technique called *global pruning* that has been developed in FPM algorithm [8], can reduces candidate sets and communication cost and increases performance.*Global pruning* is stated in [8] as follows:

Global pruning Let *X*be a candidate *k*-itemset. At each partitionD_i, $s_x(D_i) \leq s_y(D_i)$, if $Y \subset X$. Therefore the local support count of *X*is bounded by the value min $\{ s_y(D_i) \mid Y \subset X;$ and $|Y|=k\text{-}1\}$. Since the globalsupport count of *X*, $s_x(D)$, is the sum of its local support count at all the processors, the value

$smax_x(D) = \sum_{i=1}^{M} smax_x(D_i)$

where

$smax_x(D_i)$ = min $\{ s_y(D_i) \mid Y \subset X;$ and $|Y| = k\text{-}1\}$

is an upper bound of$s_x(D)$. If $smax_x(D) < minsup \times |D|$, then *X* can be pruned away.

Note that *global pruning* requires the local support counts resulted from count exchange in the previous iteration. FPM doesn't have a merger site and all processors exchange local count and so they contain all counts, but the situation is more complex in a distributed environment with merger site.DMA, DDM and ODAM algorithms haven't use *global pruning*. They have assigned *apriori_gen* function to local sites andif they intend to use *global pruning*like FPM, central site need to send all local counts to all sites and it hashuge amounts of communication overhead.Since the *apriori_gen* function has the

same process and result in all sites, we can transfer it to the merger site easily and use *global pruning.*
Note that after *global pruning*, candidate set in all nodes is similar while some items maybe doesn't exist in some node. Thus we can perform a new pruning method in NDD-FIM called *node pruning* after *global pruning* and before sending candidate itemsets to each node.

Node pruning Let $C_k$be candidate *k*-itemsets after *global pruning*. At each partitionD_i, if the value

$smax_x(D_i)$ = min { $s_y(D_i)$ | $Y \subset X$; and $|Y| = k\text{-}1$}

be equal to zero, then *X* can be pruned away from n_i.

Global pruning and *node pruning* techniques reduce candidate sets and communication cost and increase performance. Specially, high data skewness and workload balance would increase the chance of candidate set pruning [8].

4.3Reduction of idle time

Elimination of infrequent items and finding similar transaction described in section 4-1 is a time-consuming process and placing it alongside other operation is not economical. On the other hand, the processors are idle when they are waiting the results of global processingfrom center site. Also, low-speed network for transition oflocal and global results increases idle time of processors.
Furthermore, we have created two threads in local processors. First thread is for processing local results and second thread is for elimination of infrequent items.
Second thread is a dynamic background action and every time that processor is idle or wistful, it can switch to this thread.
Moreover, transition of *apriori_gen* function to the server causesthe local processors to be released from a useless and repetitive activity and to perform second thread.

4.4 The NDD-FIM algorithm

Fig.6 shows NDD-FIM's pseudocode in the local sites. It first computes support counts of 1-itemsets from local data set in the same manner as it does for the sequential Apriori, then broadcasts locally frequent items to the mergersite and starts elimination thread. Actually, at first pass, elimination thread doesn't have any infrequent items but it can find similar transactions. At next passes computed NL_k may be empty or not.
Local site stops elimination thread when receives locally large 1-itemsets in other site and candidate 2-itemsets from the merger.Subsequently, sending support counts, receiving new candidates, stopping and resuming elimination thread are performed iteratively.
In Fig. 6, $C_{k(i)}$ and $LL_{k(i)}$ are used to denote candidate and locally large *k*-itemsets at processorp_i, respectively. Also $LLO_{k(i)}$ is locally large k-itemsets at others processors exceptp_i.Fig. 7 shows NDD-FIM's pseudo code in the merger site. It receives locally frequent itemsets from local sites and computes summation of support counts. Then it finds some gl-large itemsets and sends indeterminate itemsets to other sites and receives their support counts to determine final gl-large collection.
Subsequently, the merger site executes *apriori_gen* function to generate candidate *k*-itemsets and prunes them using *global pruning* described in section 4-2. Afterwards it performs *node pruning* for each local site and broadcasts pruned candidate *k*-itemsets to all local sites. It discovers the globally frequent itemsets of that respective length after every pass.Fig. 8 shows the flow diagram of NDD-FIM algorithm.

Input: D_i,i =1, . . . , M , s_d
Output: nothing

1. k=1
2. $C_{k(i)} = I$
3. While$C_{k(i)} \neq \emptyset$
   {
4. ScanD_i to find $s_x(D_i)$for all $x \in C_{k(i)}$
5. $LL_{k(i)} = \{x \in C_{k(i)} \mid s_x(D_i) \geq s_d \times |D_i|\}$
6. Send $\{s_x(D_i) \mid x \in LL_{k(i)}\}$
7. Execute elimination thread
8. Receive $LLO_{k(i)}$
9. Send $\{s_x(D_i) \mid x \in LLO_{k(i)}\}$
10. Execute elimination thread
11. Receive $C_{k+1(i)}$
12. k=k+1
    }

Fig 6: NDD-FIM algorithm in the local sites

Input:$|D|$, s_d
Output: *L* (globally large itemsets of size 1 to *k*)

1. k=1
2. For i=1 to M
3. Receive $\{s_x(D_i) \mid x \in LL_{k(i)}\}$
4. $LL_k = \cup_{i=1}^{M} LL_{k(i)}$
5. While$LL_k \neq \emptyset$
   {
6. $s_x(D) = \sum_{i=1}^{M} s_x(D_i)$, for all $x \in LL_k$
7. $GL_k = \{x \in LL_k \mid s_x(D) \geq s_d \times |D|\}$
8. For i=1 to M
{
9. $LLO_{k(i)} = \{ x \mid x \in LL_k \wedge x \notin GL_k \wedge x \notin LL_{k(i)}\}$
10. Send $LLO_{k(i)}$
11. Receive $\{s_x(D_i) \mid x \in LLO_{k(i)}\}$
    }
12. $LLO_k = \cup_{i=1}^{M} LLO_{k(i)}$
13. $s_x(D) = s_x(D) + \sum_{i=1}^{M} s_x(D_i)$, for all $x \in LLO_k$
14. $GL_k = GL_k \cup \{x \in LLO_k \mid s_x(D) \geq s_d \times |D|\}$
15. $L = L \cup GL_k$
16. $C_{k+1} = \cup_{i=1}^{M} apriori\_gen(GL_{k(i)})$
17. Prune C_{k+1} by global pruning
18. For i=1 to M
    {
19. $C_{k+1(i)} = C_{k+1}$
20. Prune $C_{k+1(i)}$ by node pruning
21. Send $C_{k+1(i)}$
22. Receive $\{s_x(D_i) \mid x \in LL_{k+1(i)}\}$
    }
23. $LL_{k+1} = \cup_{i=1}^{M} LL_{k+1(i)}$
24. k=k+1
    }
25. Return *L*

Fig 7: NDD-FIM algorithm in the merger site

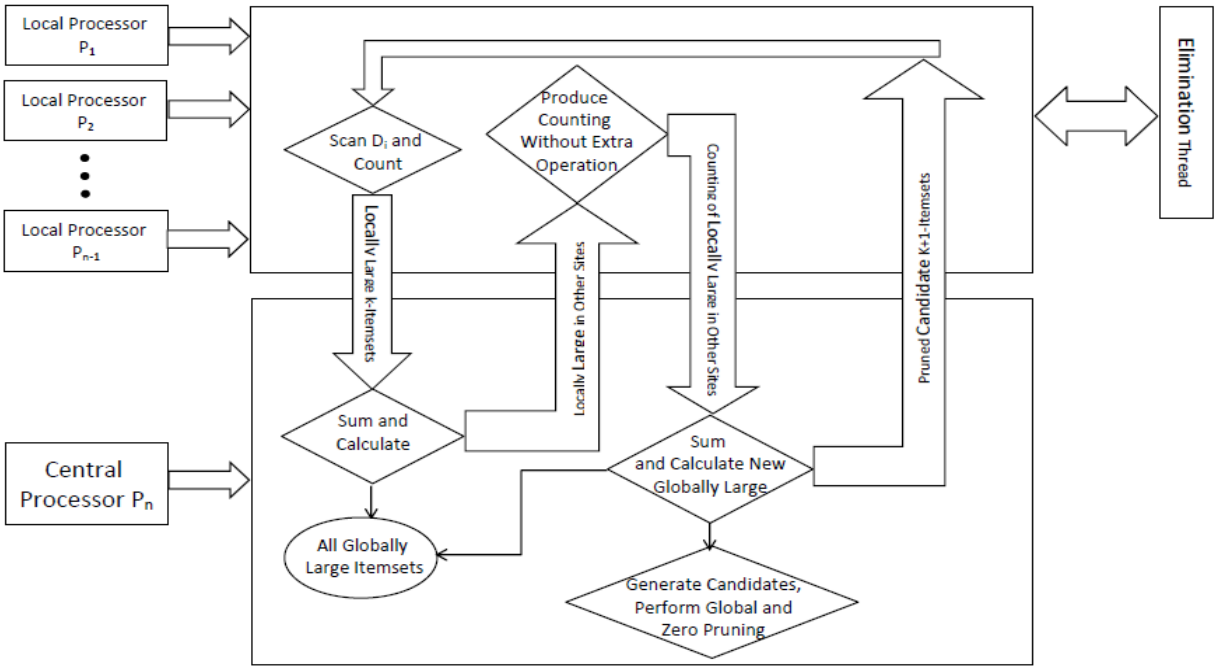


Fig 8: The flow diagram of NDD-FIM algorithm

5. IMPLEMENTATION & EXPERIMENTAL RESULTS

The proposed algorithm was implemented using Visual Studio .Net 2008 and C# language and generated all frequent itemsets satisfying the required minimum support indicated by the user. It was implemented using an Ethernet LAN consisting of 7 workstations and one merger site. The configuration of each workstation on the network was an AMD Athlon XP 2800+ with 2 GB of RAM. Also, the operating system was Windows XP Professional SP3.

The processors were interconnected via a 10/100 Mbps switch. The Message Passing Interface for .Net (MPI.Net) was used for communications.

We chose two datasets to test the communication cost of NDD-FIM versus ODAM algorithm. The datasets are taken from the FIM dataset repository page (http://fimi.ua.ac.be). The Connect dataset produces many long frequent itemsets even for high support and is typical of dense datasets. Meanwhile, Mushroom is sparse and uses low support thresholds to generate frequent itemsets. Table 5 shows the characteristics of experimental datasets.

Table 5.The characteristics of experimental datasets

Datasets	#Items	#Records	Avg.length	Type
Mushroom	119	8,124	23	sparse
Connect-4	129	67,557	43	dense

We also used data from the KDD Cup 2000 [30] to generate the data used in some experiments and test the performance of NDD-FIM versus Prefix, DDRM, and ODAM algorithms. This data set was based on click-stream data obtained from a web store called Gazelle.com. The size of the data was 4.8 Mbytes with 3,465 transactions. It included 220 attributes of customer information such as gender, occupation, age, marital status, estimated income, home market value, US State, Email, etc.We selected some attributes as categories to be investigated. Each of these categories was further subdivided into specific items, with each item being assigned an integer value used to represent it in the data file.

5.1 Communication cost experiment

To compare the number of messages that NDD-FIM and ODAM exchange among sites to generate the globally frequent itemsets in a distributed environment, we partitioned the Connect and Mushroom data sets into five partitions. Therefore, each site contains approximately 20 percent of the original data set's transactions.

Fig. 9 and 10 depicts the total size of messages that ODAM and our algorithm transmit with different support values.

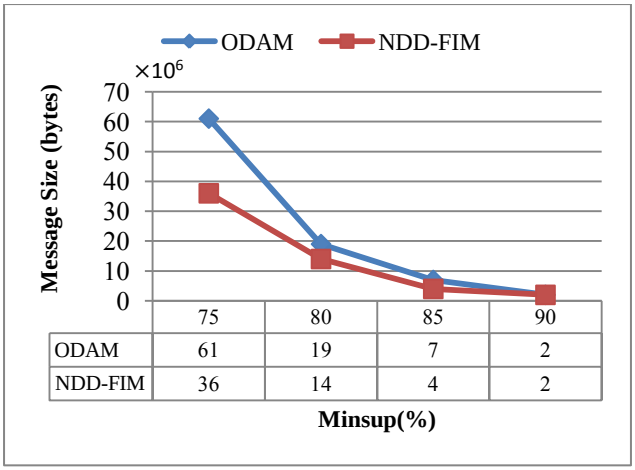
As Fig. 9 and 10 shows, proposed algorithm exchanges fewer messages among sites because of *global pruning* and *nodepruning* techniques and elimination of infrequent items in all iterations.

5.2 Execution time experiment

We conducted experiments on a set of data from KDD data set with 30 attributes where we vary the support from 4% to 10% for NDD-FIM, Prefix, DDRM, and ODAM algorithms. The data set was partitioned into 4 parts based on the number of processors. The obtained execution times of this experiment are shown in Fig. 10.

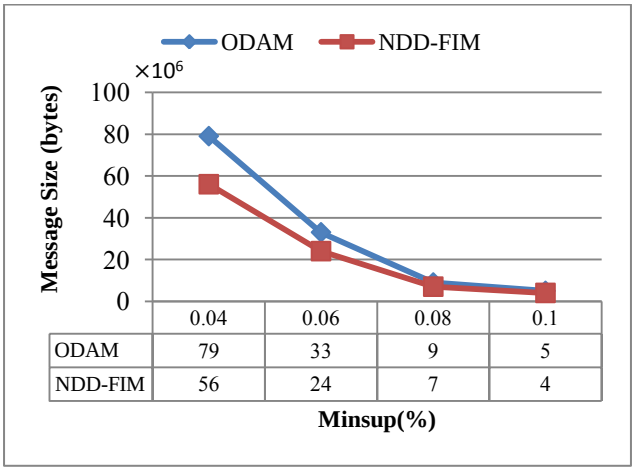
5.3 Transaction width experiment

Fig. 11 shows the results of our experiment to determine the impact of varying the transaction width on the execution time. The number of selected attributes was varied from 10 to 50 for the five sub data sets from KDD that were used in this experiment. The number of processors and the support threshold were 7 and 8%, respectively. It can be seen from Fig. 11 that as the transaction size is increased that there is a corresponding increase in the processing time. Our algorithm is able to process these transactions in a shorter time than the other algorithms.



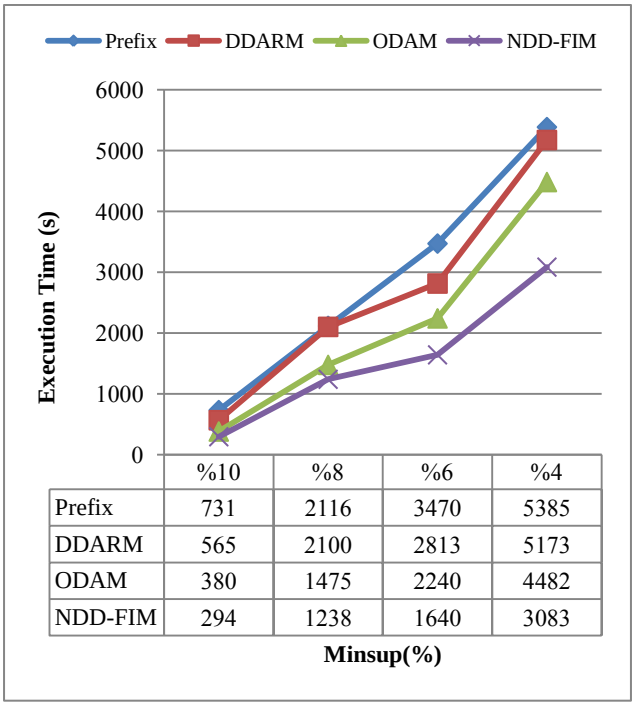
	75	80	85	90
ODAM	61	19	7	2
NDD-FIM	36	14	4	2

Fig 9: Total exchanged messages for Connect-4 dataset



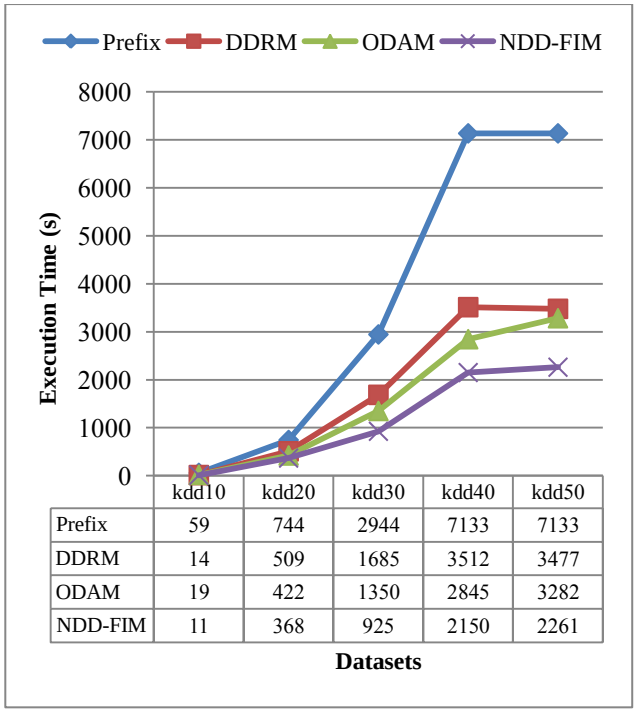
	0.04	0.06	0.08	0.1
ODAM	79	33	9	5
NDD-FIM	56	24	7	4

Fig 10: Total exchanged messages for Mushroom dataset



	%10	%8	%6	%4
Prefix	731	2116	3470	5385
DDARM	565	2100	2813	5173
ODAM	380	1475	2240	4482
NDD-FIM	294	1238	1640	3083

Fig 11: Execution time for KDD dataset



	kdd10	kdd20	kdd30	kdd40	kdd50
Prefix	59	744	2944	7133	7133
DDRM	14	509	1685	3512	3477
ODAM	19	422	1350	2845	3282
NDD-FIM	11	368	925	2150	2261

Fig 12: Execution time for various size of KDD dataset

6. CONCLUSION

Distributed Data Mining (DDM) enables learning over huge amounts of data that are situated at different geographical locations. It supports several interesting applications, ranging from fraud and intrusion detection, to market basket analysis over a wide area, to knowledge discovery from remote sensing data around the globe. Frequent Itemsets Mining (FIM) in a huge database is a worthwhile research topic in DDM area and is, however, very time consuming process. Parallel and distributed computation strategies provide suitable solutions to this problem.

In this article, the New Dynamic Distributed Algorithm for FIM (NDD-FIM) is proposed. It eliminates all infrequent items after every pass and finds more identical transactions. Furthermore, NDD-FIM can effectively reduce the required scan iterations to a database and accelerate the calculation of itemsets. Also the elimination is a dynamic background activity to fill idle time of processors. On the other hand, NDD-FIM uses local and global pruning and a merger site to reduce the communication overhead. Experimental results show that NDD-FIM achieves better than some pervious works.

7. ACKNOWLEDGMENTS

We would like to thank Blue Martini Software for contributing the KDD Cup 2000 data.

8. REFERENCES

[1] Ailing,W.2011. An Improved Distributed Mining Algorithm of Association Rules, JCIT: Journal of Convergence Information Technology, 6(4) 118-122.

[2] Roy, S., Bhattacharyya, D.K.2008. OPAM: An Efficient One Pass Association Mining Technique without Candidate Generation, JCIT: Journal of Convergence Information Technology, 3(3) 32-38.

[3] Li, Y., Sun, L., Yin, J., Bao, W., Gu,M.2010. Multi-Level Weighted Sequential Pattern Mining Based on Prime Encoding, JDCTA: International Journal of Digital Content Technology and its Applications, 4(9) 8-16.

[4] Lin, F., Le, W.,Bocor, J. 2010. Research on Maximal Frequent Pattern Outlier Factor for Online High-Dimensional Time-Series Outlier Detection, JCIT: Journal of Convergence Information Technology, 5(10) 66-71.

[5] Agrawal, R., Shafer,J.C. 1996. Parallel mining of association rules, IEEE Transactions on Knowledge and Data Engineering, 8(6) 962-969.

[6] Cheung, D.W., Han, J., Ng, V.T., Fu, A.W., Fu, Y. 1996. A fast distributed algorithm for mining association rules, In Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems, 31-42.

[7] Schuster, A., Wolff, R., Trock,D. 2005. A high-performance distributed algorithm for mining association rules, Knowledge Information System, 7(4) 458–475.

[8] Cheung, D., Xiao, Y. 1998. Effect of data skewness in parallel mining of association rules, In 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Melbourne, Australia, April,48-60**.**

[9] Schuster, A., Wolff, R. 2001. Communication-efficient distributed mining of association rules, In Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, California, May,473-484.

[10] Agrawal, R.,Srikant, R. 1994. Fast algorithms for mining association rules, In Proceedings of the 20th International Conference on Very Large Databases (VLDB94), Santiago, Chile, September,487-499.

[11] Park,J.S., Chen,M., Yu,P.S. 1995. An effective hash-based algorithm for mining association rules, In Proceedings of ACM SIGMOD International Conference on Management of Data, San Jose, California, May,175-186.

[12] Park, J.S., Chen, M., Yu, P.S. 1995. Efficient parallel data mining for association rules, in Proceedings of ACM International Conference on Information and Knowledge Management, Baltimore, MD, November,31-36.

[13] Pramudiono, I., Kitsuregawa, M. 2003. Parallel FP-Growth on PC cluster, In Proceedings of the 7th Pacific–Asia Conference of Knowledge Discovery and Data Mining, (PAKDD03)467- 473.

[14] Han, J., Pei, J., Yin, Y. 1999. Mining frequent patterns without candidate generation, Technical Report, Simon Fraser University, October,99-102.

[15] Ashrafi, M.Z., Taniar, D., Smith, K.A. 2004. ODAM: an optimized distributed association rule mining algorithm, IEEE Distributed Systems Online, 5(3).

[16] Zaki, M.J. 2000b. Parallel and distributed data mining: An introduction, In M.J. Zaki, C. Ho, (Eds.), Large-Scale Parallel Data Mining. New York, NY: Springer-Verlag. 1-23.

[17] Agrawal, R., Imielinski, T., Swami, A. 1993. Mining association rules between sets of items in large databases, In Proceedings of ACM SIGMOD International Conference on Management of Data, 207-216.

[18] Chung, S.M.,Congnan, L. 2008. Efficient mining of maximal frequent itemsets from databases on a cluster of workstations, Knowledge Information System, 16(3) 359–391.

[19] Congnan, L., Chung, S.M. 2008. A scalable algorithm for mining maximal frequent sequences using a sample, Knowledge Information System, 15(2) 149–179.

[20] Lian, W., Cheung, D.W., Yiu,S.M. 2007. Maintenance of Maximal Frequent Itemsets in Large Databases, In Proceedings of 2007 ACM Symposium on Applied Computing (SAC'07), Seoul, 388-392.

[21] Tsoumakas,G., Vlahavas, I. 2009. Distributed Data Mining, Database Technologies: Concepts, Methodologies, Tools, and Applications,Page-710.

[22] Toivonen, H. 1996. Sampling large databases for association rules, In Proceedings of 22th International Conference on Very Large Data Bases (VLDB'96), Bombay, India, 134-145.

[23] Brin, S.,Motwani,R., Ullman, J.D., Tsur, S.1997. Dynamic itemset counting and implication rules for market basket data, In Proceedings of ACM SIGMOD International Conference on Management of Data, 255-264.

[24] Ye, Y., Chiang, C.C.2006. A parallel apriori algorithm for frequent itemsets mining, In Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications,87–94.

[25] Bodon, F. 2003. A fast apriori implementation, In Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations.

[26] Wu, J., Li,X.M.2008. An efficient association rule mining algorithm in distributed database, In International Workshop on Knowledge Discovery and Data Mining (WKDD),108–113.

[27] Wessel, T.2009. Parallel mining of association rules using a lattice based approach [dissertation], Nova Southeastern University.

[28] Aggarwal, C.C., Yu, P.S.2001. A new approach to online generation of association rules, IEEE Transactions Knowledge and Data Engineering, 13(4)527-540.

[29] Cheung, D.W., Ng, V.T., Fu, A.W., Fu, Y. 1996. Efficient mining of association rules in distributed databases, IEEE Transactions on Knowledge and Data Engineering, 8(6) 911-922.

[30] Kohavi, R., Bradley, C.E., Frasca, B., Mason, L., Zheng, Z. 2000. KDD-Cup 2000 organizers Report: Peeling the Onion. SIGKDD Exploration 2(2) 86–93.

[31] Zaki, M.J. 2000c. Hierarchical parallel algorithms for association mining, In Kargupta,H & Chan P. (Eds.), Advances in Distributed and Parallel Knowledge Discovery, Cambridge, MA: MIT Press 339-336.