An Efficient Frequent ItemSets Mining Algorithm for Distributed Databases

Azam Adelpoor, Mohammad Saniee Abadeh

Abstract**—**Association Rules Mining (ARM) in large transactional databases is an important problem in the field of knowledge discovery. It has many applications in decision support and marketing strategy. Centralized and Distributed Association Rules Mining (DARM) include two phases of frequent itemset extraction and strong rule extraction. 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 Efficient Frequent ItemSets Mining for Distributed Databases (EDFIM). EDFIM has a merger site to reduce communication overhead and eliminates size of dataset partitions. The experimental results show that our algorithm generates support counts of candidate itemsets quicker than other DARM algorithms and reduces the size of average transactions, datasets, and message exchanges.

Keywords**—**Distributed Data Mining, Frequent ItemSets, Association Rule, Apriori Algorithm.

I. 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. An association 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. Many centralized 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.

Basically, frequent itemsets generation algorithms search the dataset to determine which combination of items occurs together frequently. For a fixed threshold support s, the algorithm determines which sets of items, of a given size k, are contained in at least s of the t transactions.

Azam. Adelpoor is with the Department of Computer, Science and Research Branch, Islamic Azad University, Khouzestan, Iran (corresponding author to provide phone:+986713324975; fax:+986713323593; e-mail: a.adelpoor@khouzestan.srbiau.ac.ir).

Mohammad. Saniee Abadeh, is with the Department of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran (e-mail: saniee@modares.ac.ir).

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 distributed datasets over different geographic locations and will present a well-adapted distributed approach for this purpose, based on both analytical and experimental approaches.

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 continuum action. In the first phase, we consider only counting and a local pruning strategy. In the second phase, each node sends support counts of a collection of locally frequent itemsets to the merger node. The merger node collects frequent counts and asks other node by necessity. The overhead related to communication phase in classical approaches can then be highly reduced using a constrained collection phase 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-based distribution, we believe that the key reasoning of this study will hold for many other frequent itemsets 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.

II. 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].

In some distributed data mining approach [9,15,18], a central site exists that merges the analysis of the local database at distributed sites. 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 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 itemset *x* in *D* and D_i, respectively. For a given minimum support threshold s_d, called *minsup*, an itemset *x* is *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, the proofs can be read in [21]:

Property 2.1 A globally frequent itemset must be locally frequent in at least one node.

Property 2.2 All subsets of a globally frequent itemset are globally frequent.

III. Related Works

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 [22].

The CD (Count Distribution) and DD (Data Distribution) algorithms [5] are simple parallelization of the Apriori algorithm, and assume data sets are horizontally partitioned among different nodes and each node has a copy of candidate itemsets. CD doesn't exchange data tuples between processors, and only exchanges the counts. Each processor only needs to process the data it owns, and generates its local candidate itemsets depending on its local partition. Each node obtains global counts by exchanging local counts with all other processors. The CD's communication complexity is $O(|C_k|n^2)$ in pass *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 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 itemsets in its partition and exchanges to other nodes. Next, support counts are globally summed for those candidate itemsets which are locally frequent by at least one site. Global frequent itemsets are used to generate the next level candidates.

If the probability that an itemset has the potential to be 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 [20]. 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 algorithm; FPM (Fast Parallel Mining) [8]. It has incorporated two pruning techniques, distributed pruning and global pruning, and generates candidate itemsets less than FDM.

Another Apriori-based algorithm, the Optimized Distributed Association rules Mining (ODAM), is proposed in [15]. It 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.

In [18] a Dynamic Distributed Rule Mining (DDRM) is implemented that is a dynamic extension of Prefix-based [19] algorithm and has used a lattice-theoretic approach for mining association rules. Actually DDRM partitions 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. Transferring Tid-list between local nodes and the controller node has huge amounts of overhead.

IV. Description of Proposed Algorithm

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

A. 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. 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 in (1).

$$NL_k = \begin{cases} i_i \mid (i_i \in I) \wedge (i_i \notin L_1) & k = 1 \\ i_i \mid (i_i \in I) \wedge (i_i \notin NL_{k-1}) \wedge (\nexists\, l \in L_k \mid i_i \subset l) & k \neq 1 \end{cases} \quad (1)$$

Consider the sample dataset in TABLE I 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 and the results are shown in TABLE II.

TABLE I
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 II
SAMPLE DATASET AFTER ELIMINATION OF I_5 AND I_6

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. Also, at *k*th iteration, transactions that are smaller than or equal to *k* can be eliminated. The results of these two steps are shown in TABLE III and Table IV.

This manner is effective on final iterations of mining and real world datasets which variation of transactions′s size is tremendous.

TABLE III
SAMPLE DATASET AFTER ELIMINATION OF I_4

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 IV
SAMPLE DATASET AFTER REDUCTION OF SHORT TRANSACTION

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

B. Reduction of Communication Cost

As mentioned before, in some approaches [9,15,18], the analysis of the local database at distributed sites is transmitted to a central site, and the integration is performed. In this situation, the communication complexity is $O(|C_k|n)$ in pass *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 and the communication complexity is $O(|C_k|n^2)$. Futhermore, we have used a merger site in EDFIM 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 partition D_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 global support 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

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

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 candidate generation function to local sites and if they intend to use *global pruning* like FPM, central site need to send all local counts to all sites and it has huge amounts of communication overhead. Since the candidate generation 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 EDFIM 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 partition D_i, if the value

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

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.

C. The EDFIM algorithm

Fig. 1 shows EDFIM's pseudo code in the local sites. It first eliminates infrequent items and 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 merger site. Actually, at first pass, elimination process doesn't have any infrequent items but it can find similar transactions. At next passes computed NL_k may be empty or not.

Subsequently, local site receives locally large 1-itemsets in other site and candidate 2-itemsets from the merger site. Sending support counts, receiving new candidates and elimination process are performed iteratively.

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

1. k=1
2. $C_{k(i)} = I$
3. $NL_k = \varnothing$
4. While $C_{k(i)} \neq \varnothing$
5. {
6. Elimination of infrequent items in NL_k
7. Scan D_i to find $s_x(D_i)$ for all $x \in C_{k(i)}$
8. $LL_{k(i)} = \{x \in C_{k(i)} \mid s_x(D_i) \geq s_d \times |D_i|\}$
9. Send $\{s_x(D_i) \mid x \in LL_{k(i)}\}$
10. Receive $LLO_{k(i)}$
11. Send $\{s_x(D_i) \mid x \in LLO_{k(i)}\}$
12. Receive $C_{k+1(i)}$
13. k=k+1
14. NL_k= Find infrequent items
15. }

Fig. 1 NDD-FIM algorithm in the local sites

In Fig. 1, $C_{k(i)}$ and $LL_{k(i)}$ are used to denote candidate and locally large *k*-itemsets at processor p_i, respectively. Also $LLO_{k(i)}$ is locally large k-itemsets at others processors except p_i.

Fig. 2 shows EDFIM'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.

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 $LL_k \neq \varnothing$
6. {
7. $s_x(D) = \sum_{i=1}^{M} s_x(D_i)$, for all $x \in LL_k$
8. $GL_k = \{x \in LL_k \mid s_x(D) \geq s_d \times |D|\}$
9. For i=1 to M
10. { $LLO_{k(i)} = \{ x \mid x \in LL_k \wedge x \notin GL_k \wedge x \notin LL_{k(i)}\}$
11. Send $LLO_{k(i)}$
12. Receive $\{s_x(D_i) \mid x \in LLO_{k(i)}\}$
13. }
14. $LLO_k = \cup_{i=1}^{M} LLO_{k(i)}$
15. $s_x(D) = s_x(D) + \sum_{i=1}^{M} s_x(D_i)$, for all $x \in LLO_k$
16. $GL_k = GL_k \cup \{x \in LLO_k \mid s_x(D) \geq s_d \times |D|\}$
17. $L = L \cup GL_k$
18. $C_{k+1} = \cup_{i=1}^{M} apriori\_gen(GL_{k(i)})$
19. Prune C_{k+1} by global pruning
20. For i=1 to M
21. { $C_{k+1(i)} = C_{k+1}$
22. Prune $C_{k+1(i)}$ by node pruning
23. Send $C_{k+1(i)}$
24. Receive $\{s_x(D_i) \mid x \in LL_{k+1(i)}\}$
25. }
26. $LL_{k+1} = \cup_{i=1}^{M} LL_{k+1(i)}$
27. k=k+1
28. }
29. Return L

Fig. 2 NDD-FIM algorithm in the merger site

V. IMPLIMENTATION AND RESULTS

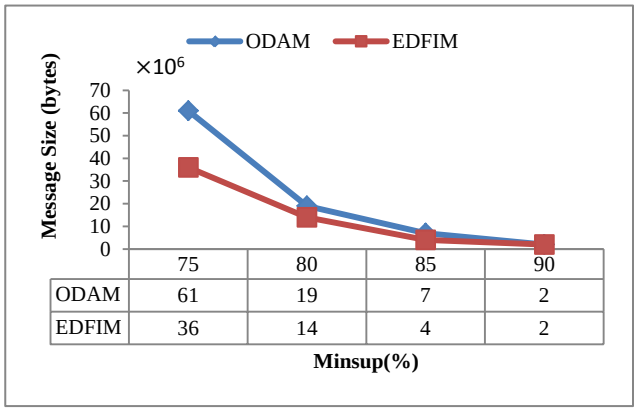
The proposed algorithm was implemented using Visual Studio .Net 2008 and C# language and generated all frequent itemsets satisfying the required minimum support indicated by the user. It was implemented using an Ethernet LAN consisting of 7 workstations and one merger site. The configuration of each 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, Connect-4 and Mushroom, to test the communication cost of EDFIM 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.

We also used data from the KDD Cup 2000 [16] to generate the data used in some experiments and test the performance of EDFIM versus Prefix, DDARM, 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 and included 220 attributes of customer information. We selected some attributes as categories to be investigated.

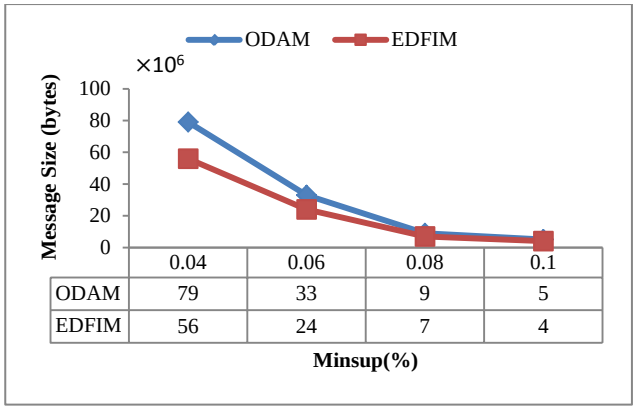
A. Communication cost experiment

To compare the number of messages that EDFIM and ODAM exchange among, we partitioned the Connect and Mushroom data sets into five partitions. Fig. 3 and 4 depicts the total size of messages that ODAM and our algorithm transmit with different support values. As figures show, proposed algorithm exchanges fewer messages among sites because of *global pruning* and *node pruning* techniques and elimination of infrequent items in all iterations.



	75	80	85	90
ODAM	61	19	7	2
EDFIM	36	14	4	2

Fig. 3 Total exchanged messages for Connect-4 dataset

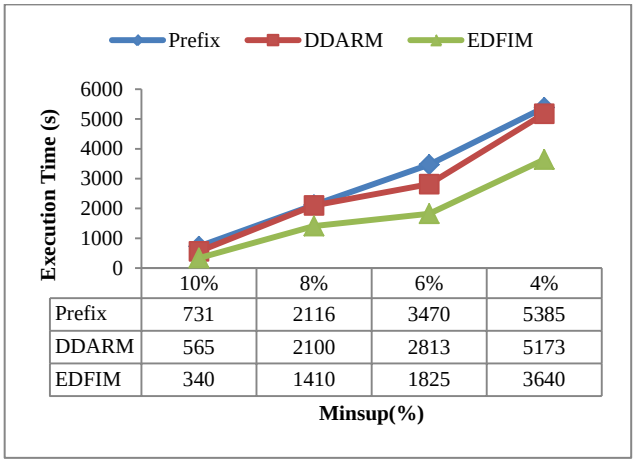


	0.04	0.06	0.08	0.1
ODAM	79	33	9	5
EDFIM	56	24	7	4

Fig. 4 Total exchanged messages for Mushroom dataset

B. 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 EDFIM, Prefix, and DDARM 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. 5.

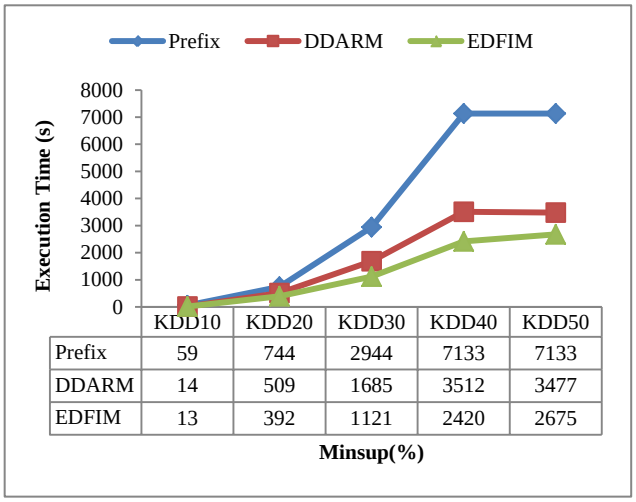


	10%	8%	6%	4%
Prefix	731	2116	3470	5385
DDARM	565	2100	2813	5173
EDFIM	340	1410	1825	3640

Fig. 5 Execution time for KDD dataset

C. Transaction width experiment

Fig. 6 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. The number of processors and the support threshold were 7 and 8%, respectively. It can be seen from Fig. 6 that our algorithm is able to process these transactions in a shorter time than the other algorithms.



	KDD10	KDD20	KDD30	KDD40	KDD50
Prefix	59	744	2944	7133	7133
DDARM	14	509	1685	3512	3477
EDFIM	13	392	1121	2420	2675

Fig. 6 Execution time for various size of KDD dataset

VI. CONCLUSION

Distributed Data Mining (DDM) enables learning over huge amounts of data that are situated at different geographical locations. Frequent Itemsets Mining (FIM) is a part of DDM, and is a worthwhile research topic. However, it is a very time consuming process and many Parallel and distributed computation strategies provide suitable solutions to this problem.

In this article, an Efficient Distributed Algorithm for FIM (EDFIM) is proposed. It eliminates all infrequent items after every pass and finds more identical transactions. Furthermore, EDFIM can effectively reduce the required scan iterations to a database and accelerate the calculation of itemsets. On the

other hand, EDFIM uses local and global pruning and a merger site to reduce the communication overhead. Experimental results show that EDFIM achieves better than some pervious works.

ACKNOWLEDGMENT

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

REFERENCES

[1] W. Ailing, "An Improved Distributed Mining Algorithm of Association Rules", JCIT: Journal of Convergence Information Technology, vol 6, no 4, pp. 118-122, 2011.

[2] S. Roy, D.K. Bhattacharyya, "OPAM: An Efficient One Pass Association Mining Technique without Candidate Generation", JCIT: Journal of Convergence Information Technology, vol 3, no 3, pp. 32-38, 2008.

[3] Y. Li, L. Sun, J. Yin, W. Bao, M. Gu, "Multi-Level Weighted Sequential Pattern Mining Based on Prime Encoding", JDCTA: International Journal of Digital Content Technology and its Applications, vol 4, no 9, pp. 8-16, 2011.

[4] F. Lin, W. Le, J. Bocor responding author, "Research on Maximal Frequent Pattern Outlier Factor for Online High-Dimensional Time-Series Outlier Detection", JCIT: Journal of Convergence Information Technology, vol 5, no 10, pp. 66-71, 2010.

[5] R. Agrawal, J.C. Shafer, "Parallel mining of association rules", IEEE Transactions on Knowledge and Data Engineering, vol 8, no 6, pp. 962-969, 1996.

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

[7] A. Schuster, R. Wolff, D. Trock, "A high-performance distributed algorithm for mining association rules", Knowledge Information System, vol 7, no 4, pp. 458–475, 2005.

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

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

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

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

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

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

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

[15] M.Z. Ashrafi, D. Taniar, K.A. Smith, "ODAM: an optimized distributed association rule mining algorithm", IEEE Distributed Systems Online, vol 5, no 3, 2004.

[16] R. Kohavi, C.E. Bradley, B. Frasca, L. Mason, Z. Zheng, "KDD-Cup 2000 organizers" Report: Peeling the Onion. SIGKDD Exploration, vol 2, no 2, pp. 86–93, 2000.

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

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

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

[20] D.W. Cheung, V.T. Ng, A.W. Fu, Y. Fu, "Efficient mining of association rules in distributed databases", IEEE Transactions on Knowledge and Data Engineering, vol 8, no 6, pp. 911-922, 1996.

[21] L. Aouad, N. Khac, T. Kechadi, "Performance study of distributed Apriori-like frequent item sets mining", Knowledge Information System, no 23, pp. 55-72, 2010.

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