Distributed Frequent Itemset Mining using Trie Data Structure

E. Ansari, G.H. Dastghaibifard, M. Keshtkaran, H.Kaabi

Abstract— **Finding association rules is one of the most investigated fields of data mining. Computation and communication are two important factors in distributed association rule mining. In this problem Association rules are generated by first mining of frequent itemsets in distributed data. In this paper we proposed a new distributed trie-based algorithm (DTFIM) to find frequent itemsets. This algorithm is proposed for a multi-computer environment. In second phase we added an idea from FDM algorithm for candidate generation step. Experimental evaluations on different sort of distributed data show the effect of using this algorithm and adopted techniques.**

Index Terms—**Frequent itemset mining, FDM, Trie**

I. Introduction

The association rule mining (ARM) is very important task within the area of data mining [1]. Given a set of transactions, where each transaction is a set of literals (called items), an association rule is an expression of the form X ➔Y, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions of the database which contain X tend to contain Y. An example of an association rule is: "30% of transactions that contain beer also contain diapers; 2% of all transactions contain both of these items". Here 30% is called the confidence of the rule, and 2% the support of the rule. The problem is to find all association rules that satisfy user-specified minimum support and minimum confidence constraints. Frequent patterns discovered via mining processes not only themselves are interesting, but also are useful to other data analysis and mining tasks, including associative classification, clustering, cube computation and analysis, and gradient mining and multi-dimensional discriminant analysis [19].

The main task of every ARM algorithm is to discover the sets of items that frequently appear together, the frequent itemsets. Finding frequent itemsets in transaction databases has been demonstrated to be useful in several business applications [14].

Manuscript received July 22, 2008.

Ebrahim Ansari Chelche is Msc Student of Computer Science and Engineering Department, Shiraz University, Shiraz, Iran. (email: ansari@cse.shirazu.ac.ir)

G.H. Dastghaibifard is assistant professor of Computer Science and Engineering Department, Shiraz University, Shiraz, Iran. (email: dstghaib@shirazu.ac.ir)

Morteza Keshtkaran is Msc Student of Computer Science and Engineering Department, Shiraz University, Shiraz, Iran. (email: mkeshtkaran@cse.shirazu.ac.ir)

Hani Kaabi. (email: hani@sei.ir)

Hipp et. al. [16] provides a general survey on efficient mining of association rules in transaction and/or relational databases. AIS [17], SETM [18] and Apriori [2] can be considered as the first generation of association rule mining algorithms. Apriori algorithm is by far the most well-known association rule mining algorithm. AprioriTID [2] is an extension of the basic Apriori approach. Instead of relying on the raw database, AprioriTID internally represents each transaction by current candidates it contains. In AprioriHybrid both Apriori and AprioriTID approaches are combined [2].

Many algorithms have been proposed to find frequent itemsets from a very large database. The number of database scans required for the task has been reduced from a number equal to the size of the largest itemset in Apriori [2], to typically just a single scan in modern ARM algorithms such as Sampling and DIC [3, 4]. Efficient mining of association rules in transaction And/or relational databases has been studied substantially. [2]- [6]

When data is saved in a distributed database, a distributed data mining algorithm is needed to mine association rules. Mining association rules in distributed environment is a distributed problem and must be performed using a distributed algorithm that doesn't need raw data exchange between participating sites. Distributed association rules mining (DARM), has been addressed by some researches and number of distributed algorithms have been proposed [9]-[13].

Apriori [2] is one of the most popular data mining approaches for finding frequent itemsets from transactional datasets. The Apriori algorithm is the main basis of many other well-known algorithms and implementations. The main challenge faced by the researchers in frequent itemset mining has been to reduce the execution time. One of the best implementation of apriori algorithm is published by Bodon [1]. We use Bodon sequential idea to provide a distributed algorithm. The main reason we adopted Bodon's implementation for parallel computing is because Bodon's implementation using the trie data structure outperforms the other implementations using hash tree [6]-[8]. In this algorithm a Trie-based structure has been used and has been constructed in each local site that at the end of each iteration all local sites synchronized their Tries. This algorithm called DTFIM (Distributed Trie-based frequent itemset mining).

In next step, one of the ideas proposed in FDM [10] has been used to improve our implementation. This part has been embedded to main algorithm. The more skewed distributed database, the more efficiency of the technique is.

The rest of the paper is organized as follows. Section 2

introduces related work on frequent itemsets mining. Section 3 presents our implementation for DTFIM and revised DTFIM. Section 4 presents the experimental results of our implementation on a multi computer environment. Section 5 concludes the paper.

A. Notation and Problem Definition

Let $I = \{i_1, i_2, \dots, i_m\}$ be the items in a certain domain. An *itemset* is a subset of *I*. A *k*-itemset is an itemset with *k* items from *I*. A database *DB* is a list of *transactions* where each transaction *T* is also a subset of *I*.

Now assume that there are *n* sites $S_1, S_2, \dots, S_n$ in a distributed system, which communicate by message passing. Let $\overline{DB} = \{DB^1, DB^2, \dots, DB^n\}$ be a "horizontal" partition of *DB* into *n* parts (where $\bigcap_{i=1}^{n} DB_i = \emptyset$ and $\bigcup_{i=1}^{n} DB_i = DB$). We allocate each DB^i to the site S_i.

For any itemset *X* and transaction *T* we say *T contains X* if and only if $X \subseteq T$. For any itemset *X* and any group of transactions *A*, $Support(X, A)$ is the number of transactions in *A* which contain *X*. We call $Support(X, DB)$ the *global support count* of the itemset *X* in the database *DB* and $Support(X, DB^i)$ the *local support count* of *X* at site S_i. For a given minimum support threshold *s*, *X* is *globally large* (or *globally frequent*) if $Support(X, DB) \geq s \times D$, where *D* is the number of transactions in database *DB*; correspondingly *X* is *locally large* (or *locally frequent*) at site S_i if $Support(X, DB^i) \geq s \times D_i$, where D_i is the number of transactions in database partition DB_i. In the following, *L* denotes the globally large itemsets in *DB*, and $L_{(k)}$ the globally large *k*-itemsets in *L*. The essential task of a distributed association rule mining algorithm is to find the globally large itemsets *L*.

II. Previous Work

Since its introduction in 1993 [1], many algorithms with different approaches have been suggested for the ARM problem. Here we review some of the related work that form a basis for our algorithm.

A. The Apriori Algorithm

The Apriori algorithm is proposed by Agrewal in [2] and is the basis for many other FIM algorithms.
In the first pass, the occurrences of each item is being counted and the items with insufficient support get removed to create $L_{(1)}$, the collection of large 1-itemsets.
A subsequent pass, say pass *k*, consists of two steps:

- The large *(k-1)*-itemsets collection $L_{(k-1)}$ found in the previous pass is used to generate C_k, the list of candidate *k*-itemsets; which is a superset of the set of all large *k*-itemsets. A candidate is generated from every two large *(k-1)*-itemsets which are similar in their first *k*-2 items. Then, the candidates that have an infrequent subset are removed from the set of candidates.
- The database is scanned and the support count for each candidate itemset in C_k is determined. Removing items with support counts less than the minimum required gives us the large k-itemsets ($L_{(k)}$).

B. The Trie-based Apriori

Bodon shows that using efficient data structures and implementation is very important in improving the performance of Apriori algorithm [6]. He proposed a fast Apriori implementation using the trie data structure instead of a hash tree which was used in the classical approaches.

A trie is a rooted, labeled tree. In the FIM setting each label is an item. The root is defined to be at depth 0 and a node at depth d can point to nodes at depth d+1. A pointer is also referred to as edge or link. Each node represents an item sequence that is the concatenation of labels of the edges that are on the path from the root to the node. So a path from root to each node represents an itemset. In this implementation, the value of each node is the support count for the itemset it represents. In Fig.1 a trie is displayed. The path to the node representing itemset $\{A, B\}$ is shown in blue. The support count for this itemset is 7.

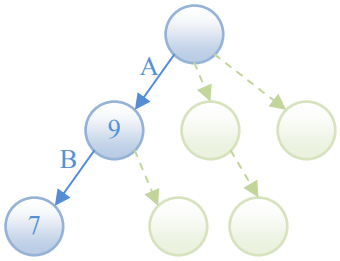

Fig.1 A sample trie

For each transaction record T in the database the trie (containing the candidate itemsets) is recursively traversed and the value of each leaf node will be incremented if T contains the itemset represented by that node. The traverse of the trie is driven by elements of T. At the end, nodes whit a support count less than the required minimum will be pruned.

In the candidate generation phase, we just need to add a leaf node to its left siblings to create new valid candidates, eliminating the need for further processing. In Fig.2 shows a trie structure before and after the new candidates are generated.

Fig.2 Candidate generation on a trie data structure

C. The FDM Algorithm

We used an idea proposed by FDM [10] to improve performance on CD [9]. If itemset *X* is globally frequent,

there is at least one site S_i in which all subsets of *X* are locally frequent. For a proof to this lemma see [10]. Therefore, if we cannot find at least one site S_i in which all subsets of *X* are locally frequent, *X* is not globally frequent and we can prune it.

III. Our Implementation

The point of this algorithm is that every site keeps a copy of trie locally, and they synchronize their data so that all local trie copies are the same at the end of each stage. After local support is counted, all sites share their support counts and determine the global support counts, in order to remove infrequent itemsets from their local trie.

A. The DTFIM Algorithm

At the beginning, each site scans its local database independently, and determines the local count of items (1-itemsets). For this purpose, a vector is used to keep count of every item. Each site reads its local transaction records one by one and increase the count of items accordingly. At the end of this stage, sites synchronize their data to determine globally large 1-itemsets ($L_{(1)}$). Using $L_{(1)}$ each site initializes its local trie copy; thus local trie copies are all alike at the end of the pass.

At second pass, the support counts for 2-itemsets shall be calculated. For this purpose a two-dimensional array is created in each site. Like Bodon, we used a triangular structure to better utilize memory. Unlike future passes, for each transaction record T, we determine all 2-itemsets that are subsets of T and increase their count. At the end, the counts are synchronized and global support count for 2-itemsets is calculated, and each site inserts large 2-itemsets into its local trie copy.

From here on, in each pass k (k≥3) a candidate large k-itemset collection (C_k) is created from the list of large (k-1)-itemsets created in the previous pass ($L_{(k-1)}$). Each site generates its own local copy of C_k, and as the results are uniform at the end of each pass, so will be the list of created candidates. As with the sequential trie-based Apriori, we add every leaf node to its left sibling nodes in the trie.

Now we must prune the infrequent k-itemsets from C_k. To reach this goal, for each transaction record T in local database, each site traverses its trie recursively and finds leaves; if T contains the k-itemset represented by a leaf, the support count on that node is increased.

Before we can prune infrequent itemsets from C_k we have to synchronize local support counts. Each site uses the same depth-first traversal algorithm to find leaves and put their values (local support counts) into a vector, ensuring uniform order in all the sites. Therefore, only the new support count values from the trie are being transferred between sites.

For the pruning step, each site does a depth-first traversal of the trie one more time, updating the value of each leaf node upon traversing it with the value from the global support counts vector and removing the itemset if its support count does not satisfy the minimum required. The output of this step will be $L_{(k)}$ and is uniform in every site.

Figure 3 shows a simple illustration of DTFIM algorithm.

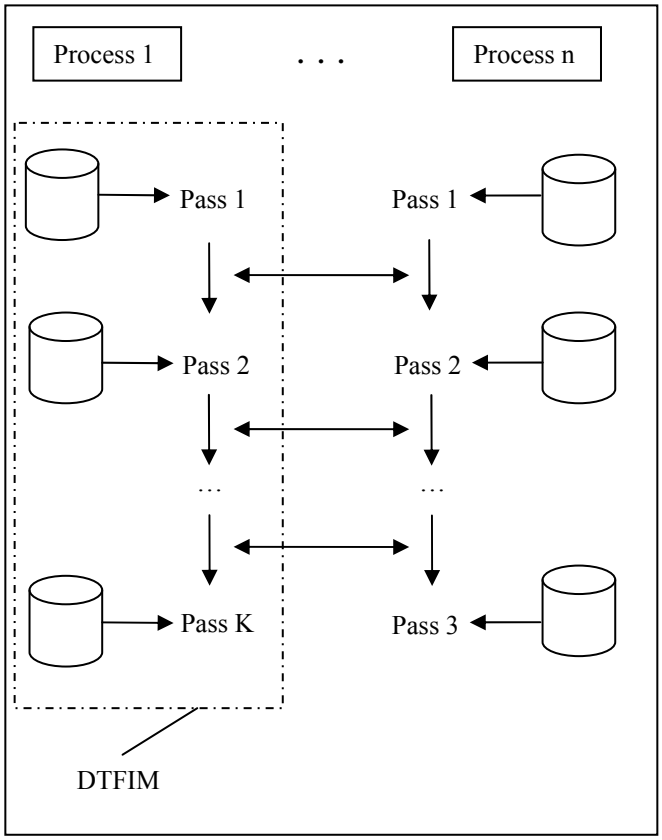

Fig.3 A Schema of DTFIM algorithm

B. The Revised DTFIM

In DTFIM algorithm, if we can predict and remove an infrequent itemset X from C_k before each site starts to calculate its local support counts, we save the time needed to process X for every T in transaction records database on every site, just to find out that X is an infrequent itemset later. One technique to predict an itemset X is infrequent is proposed by FDM and discussed in section 2.3. If X is globally frequent, there is at least one site in which all subsets of X are locally frequent.

We revised our algorithm to use this theorem, as it would not impose extra processing time and would give a considerable performance boost in most cases. We added a local frequency indicator bit vector to each trie node, each bit j of this vector signifying whether the corresponding itemset is locally large in site j or not. Each site updates its own bit at the end of every support counting step and sends it along with local support counts. At the candidate generation step we do not candidate a k-itemset if by examining the local frequency indicators of its subset (k-1)-itemsets we find out there is no site in which all of the subsets are frequent.

Fig.4 (a) shows how the local frequency indicator for an itemset is created. Each site contributes its own bit. In Fig.4 (b) local frequency indicators for all 3-itemset subsets of an example itemset {A, B, C, D} is displayed. In this case there is site 1 in which all of these subsets are frequent, so {A, B, C, D} is a candidate large itemset.

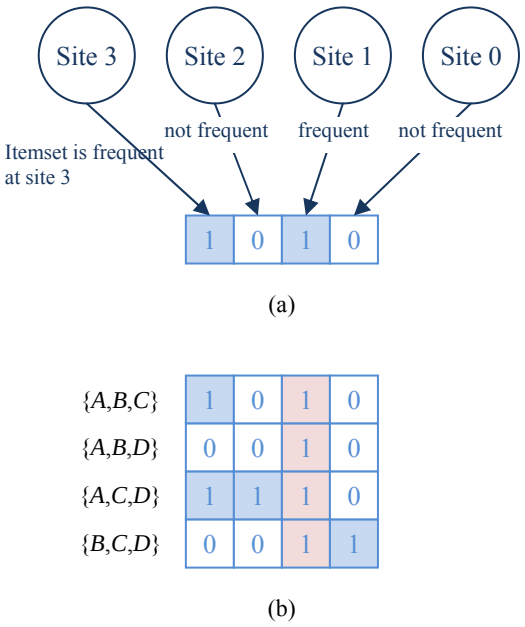

Fig.4 Local frequency indicator creation and usage

IV. Experimental Results

We have implemented all programs in C++ using Visual Studio 2005. The implementations have been tested on a workstation for which Windows XP is running on every node. This workstation consists of eight 3.4GHz Pentium IV PC with 512 MB of main memory, which are interconnected via 10M/100M hub. Parallel message passing software MPICH 2(.Net version) is used here [15]. To empirically evaluate the effect of using proposed technique several tests are performed on the datasets kosarak, accident and T40I10D100K. These datasets are available on FIM repositoryi.

Proposed algorithm is implemented and tested in our environment. The communication and computation are measured with various numbers of nodes and various minimum support values. In every experiment the original dataset is horizontally divided in a number of fragments, each of them is assigned on a node.

In Table 1, 2 and 3 experimental results for three samples database is shown. Each column shows the result of algorithm for various numbers of sites and every row illustrates one minimum support. "1 site" column represents sequential results.

In fig.5, fig.6 and fig.7, there are three experimental result diagram of our implementation in various numbers of sites to show speed up. First figure show results on *kosarak* database by support threshold equals to 0.00171. And second figure illustrate the results by database T40I10D100K and support 0.0009. Third figure, fig.7, shows result for database accident with support threshold equal to 0.49973.

V. Conclusion

Frequent itemsets mining is one of the most important areas of data mining. Bodon presented an implementation that solved frequent itemsets mining problem in most cases faster than other well-known implementations. In this paper, we used the Bodon's ideas for design an algorithm to distributed computing in a no shared memory multi computer environment. The proposed algorithm is revised with some FDM algorithm ideas. In some cases these adopted techniques causes more efficiency. Our experimental results show the efficiency of proposed algorithm. These results show Trie data structure can be used for distributed association rule mining not just for sequential algorithms.

Table 1 Execution times of DTFIM for database kosarak

minimum support	1 site	2 sites	4 sites	8 sites
0.04848	16.91	8.61	4.33	2.22
0.00303	22.38	11.31	5.8	3
0.00202	34.81	17.75	9.16	4.98
0.00171	89.25	45.6	24.49	14.13
0.000121	173.75	92.02	49.93	29.25
0.00091	552.137	287.29	165.67	101.7

Table 2 Execution times of DTFIM for database T40I10D100K

minimum support	1 site	2 sites	4 sites	8 sites
0.03	9.52	4.80	2.44	1.28
0.01	37.90	20.03	11.32	6.90
0.009	119.91	63.17	36.73	23.08
0.008	148.62	77.82	45.96	29.23
0.0058	297.91	156.90	94.17	62.01

Table 3 Execution times of DTFIM for database accident

minimum support	1 site	2 sites	4 sites	8 sites
0.61731	21.85	11.22	5.85	3.11
0.49973	33.55	19.53	11.60	7.02
0.41154	119.74	74.47	45.62	27.59

Acknowledgment

The Authors thank ITRC (Iranian Telecommunication Research Center) for their financial support. And thanks F.Alimardani for her assistance.

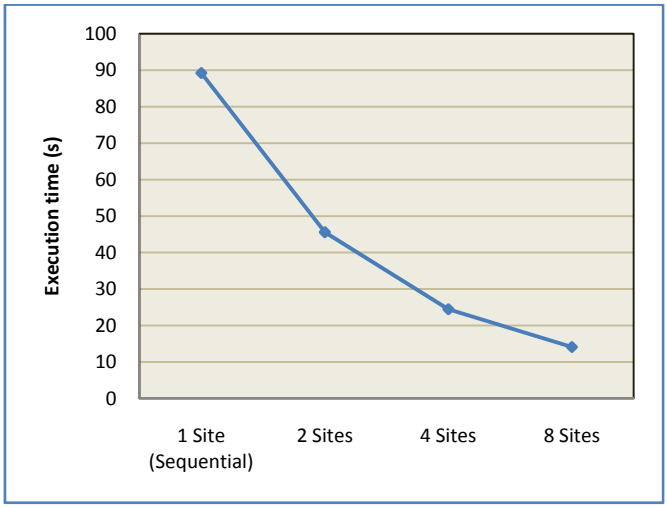

Fig.5 Execution times for database *kosarak* with minimum support threshold of 0.00171

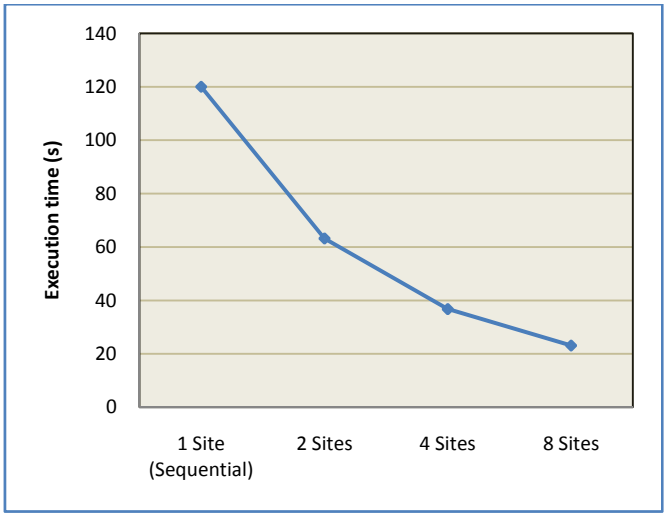

Fig.6 Execution times for database *T40I10D100K* with minimum support threshold of 0.0009

Fig.7 Execution times for database *accident* with minimum support threshold of 0.49973

REFERENCES

[1] R.Agrawal, T.Imielinski and A.Swami. Mining association rules between sets of items in large databases. In Proc. of the ACM SIG-MOD Conference on Management of Data, 1993, pp 207-216.

[2] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules, *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487-499.

[3] H. Toivonen, T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, Sampling large databases for association rules, *In Proceedings 22nd International Conference on Very Large Data Bases*, 1996, pages 134–145.

[4] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Vol.26(2) of SIGMOD Record, 1997, pp 255–264.

[5] J.Han, J.Pie,Y.Yin and R.Mao. Mining frequent pattern without candidate generation: A frequent-pattern tree approach. *Data Mining and knowledge discovery*, 2003.

[6] F. Bodon, "A Fast Apriori Implementation," In B. Goethals and M. J. Zaki, editors, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Vol. 90 of CEUR Workshop Proceedings, 2003.

[7] F. Bodon, "Surprising Results of Trie-based FIM Algorithm," In B. Goethals, M. J. Zaki, and R. Bayardo, editors, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Vol. 90 of CEUR Workshop Proceedings, 2004.

[8] F. Bodon, A Survey on Frequent Itemset Mining, Technical Report, Budapest University of Technology and Economic, 2006.

[9] R.Agrawal and J.Shafer. Parallel mining of association rules. *IEEE Transaction on Knowledge and Data Engineering*, Vol.8, No.6, 1996, pp 962-969.

[10] D. W. Cheung, and et al, A Fast Distributed Algorithm for Mining Association Rules. *In Proc. Parallel and Distributed Information Systems, IEEE CS Press*, 1996, pp 31-42.

[11] A. Schuster and R. Wolf, Communication-Efficient Distributed Mining of Association Rules, *In Proc. ACM SIGMOD International Conference on Management of Data, ACM Press*, 2001, pp 473-484.

[12] A. Schuster, R. Wolf, and D. Trock. A High-Performance Distributed Algorithm for Mining Association Rules, *Knowledge And Information Systems (KAIS) Journal*, Vo.7, No.4, 2005.

[13] M. Z Ashrafi, D. Taniar and K. Smith, ODAM: an Optimized Distributed Association Rule Mining Algorithm, *IEEE Distributed Systems Online*, Vol. 5, No.3, 2004.

[14] M. S. Chen, J. Han, and P. S. Yu, "Data Mining: An Overview from a Database Perspective, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, 1996, pp. 866-883.

[15] M. Snir, S. Otto, S. Huss-Lederman, D.Walker, J. Dongarra, *MPI: The Complete Reference*, The MIT Press, 1996

[16] J. Hipp, Ulrich Güntzer, Gholamreza Nakhaeizadeh, Algorithms for association rule mining - a general survey and comparison, ACM SIGKDD Explorations Newsletter, 2000, Volume 2, Issue 1, pages 58-64

[17] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proceedings 1993 ACM SIGMOD Intl. Conf. on Management of Data, Washington, DC, May 1993, pp. 207–216.

[18] Maurice Houtsma, Arun Swami, Set-oriented data mining in relational databases, Data & Knowledge Engineering, Volume 17, Issue 3, December 1995, Pages 245-262

[19] Jiawei Han, Hong Cheng, Dong Xin, Xifeng Yan, Frequent pattern mining: current status and future directions, Data Mining and Knowledge Discovery, Volume 15, 2007, 55-86

[i] `http://fimi.cs.helsinki.fi/data/`