World Applied Sciences Journal 9 (5): 546-552, 2010
ISSN 1818-4952
© IDOSI Publications, 2010

Using Candidate Hashing and Transaction Trimming in Distributed Frequent Itemset Mining

Ebrahim Ansari Chelche, M.H. Sadreddini and Gh. Dastghaybifard

Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran

Abstract: Frequent itemset mining is one of the most popular data mining tasks which has wide application areas and is redefined in distributed environment. Computation and communication are two important factors in distributed frequent itemset mining. In this paper two techniques have been exploited to reduce communication and improve the running time in a distributed environment. These techniques were proposed previously for centralized setting and here they are adopted in the FDM algorithm as one of the well-known distributed association rules mining algorithm. The proposed algorithm uses Trie data structure for better performance. Experimental evaluations on different sort of distributed data show the effect of using these adopted techniques.

Key words: Data mining · Association rule mining · Distributed Frequent Itemset Mining · Hash Filtering · FDM

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 [2].

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 [3].

Hipp *et al*. [4] provides a general survey on efficient mining of association rules in transaction and/or relational databases. AIS [5], SETM [6] and Apriori [7] 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 [7] 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 [].

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 [7], to typically just a single scan in modern ARM algorithms such as Sampling and DIC [8,9]. Efficient mining of association rules in transaction And/or relational databases has been studied substantially [7-11].

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

Corresponding Author: Ebrahim Ansari Chelche, Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran

association rules mining (DARM), has been addressed by some researches and number of distributed algorithms have been proposed [12-16].

Apriori [7] 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 [11]. 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 [7,11,18].

In this paper we have exploited two techniques to reduce communication and improve the run time in distributed environment. These techniques were proposed previously for centralized setting [19] and we have adopted them in the FDM [13] algorithm as one of the well-known distributed association rule mining algorithms. After implementation and running new algorithm on different datasets and comparing them by classic FDM, the result shows some optimization.

Notation and Problem Definition: Let $I = \{i_1, i_2, \ldots, i_n\}$ 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 $S_1, S_2, \ldots, S_n$ in a distributed system, which communicate by message passing. Let $DDB = \{DB^1, DB^2, \ldots, DB^n\}$ be a "horizontal" partition of DB into n parts. 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 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 if $Support(X, DB) = s \times D_i$, 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 frequent itemset mining algorithm is to find the globally large itemsets L.

Previous Works: 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 forms a basis for our algorithm.

The Apriori Algorithm: The Apriori algorithm is proposed by Agrewal in [7] 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 ($L_{(k)}$).

The Trie-based Apriori: Bodon shows that using efficient data structures and implementation is very important in improving the performance of Apriori algorithm [11]. 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 Figure. 1a trie is displayed. The path to the node representing itemset {A, B} is shown in blue. The support count for this itemset is 7.

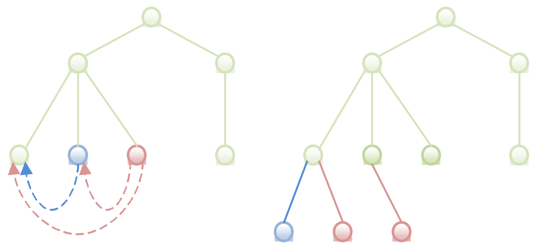
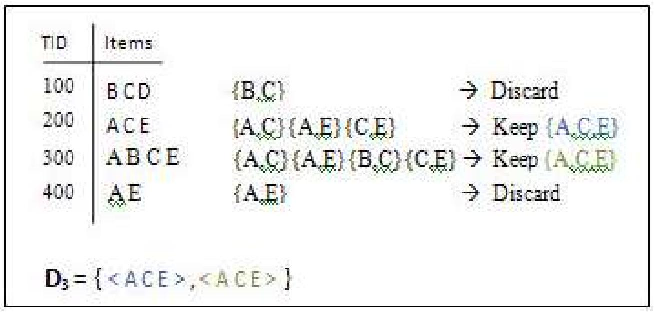


Fig. 1: Candidate generation on a trie data structure



TID	Items		
100	B C D	{B,C}	→ Discard
200	A C E	{A,C} {A,E} {C,E}	→ Keep {A,C,E}
300	A B C E	{A,C} {A,E} {B,C} {C,E}	→ Keep {A,C,E}
400	A E	{A,E}	→ Discard

D_3 = {<A C E>,<A C E>}

Fig. 2: An Example of transaction trimming

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 Figure. 2 shows a trie structure before and after the new candidates are generated.

The FDM Algorithm: In each site, FDM [13] finds the local support counts and prunes all infrequent local candidate sets. After completing local pruning, each site broadcasts messages containing all the remaining candidate sets to all other sites to request for their support counts. It then decides whether large itemsets are globally frequent and generates the candidate itemsets from those globally frequent itemsets. This process continues until no globally frequent itemsets is generated or no candidate set is produced.

To reduce message communication, algorithm uses polling site method. In this method every itemset is assign to one local site and this site must calculate support count of it. So if a site need support count of any itemset, ask from its polling site. This idea reduces communications between processes. FDM's main advantage over CD [12 is that it reduces the communication overhead.

Hash-Based Algorithm: As illustrated in section 1, in each pass we use the set of large itemset $L_{(i)}$, to create the set of candidate large itemsets $C_{(i+1)}$, by joining $L_{(i)}$ with $L_{(i)}$ *(i-1)*-common items for the next pass. We then scan the database and count the support of each itemset in $C_{(i+1)}$ in order to determine $L_{(i+1)}$. In general, more number of itemsets in $C_{(i)}$ cause higher computation time to determine $L_{(i)}$. Therefore, if the number of candidate itemsets can be reduced, the run time of frequent itemsets counting would be reduced.

As proposed in [16], the DHP algorithm uses the technique of hashing to filter out unnecessary itemset for next candidate generation. When the support of candidate k-itemset is counted by scanning the database, DHP store some information about candidate *(k+1)*-itemsets in hash table to prune unnecessary candidate itemsets. Each bucket in hash table consists of a number to represent how many itemsets have been hashed to this bucket thus far. For every candidate k-itemsets present in a transaction the count is incremented in corresponding transaction. Thus, at the end of iteration, we have an upper bound on the support count of every candidate of the next iteration. At the start of next iteration if the count

In Ordinary style:

```
Do local_ apriori( )
Create hash table( )
For each hash_bucket H
    If (count (H)>local_min_sup)
        Send polling_request (Polling_Hash(X))
Receive all hash_Counts from their polling site
FDM_Apriori( )      //do ordinary FDM on dataset
Transaction_Trimming in local database
```

In Polling site Style:

```
If receive any_hash_request H
{
    Broadcast (hash_count_request H)
    Receive Sumerize (hash_count H) From all sites
        For all hash_Bucket H
            If (count(H))>global_min_sup
                Broadcast(hash_count)
}
```

Fig. 3: Pseudo Code of algorithm

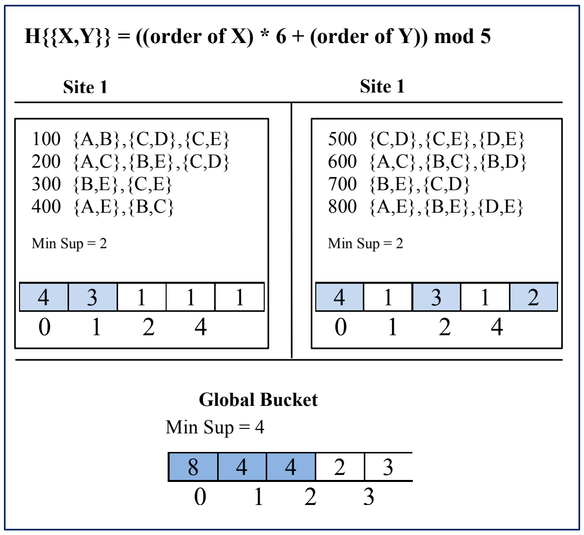


Fig. 4: An Example of Distributed Hash Filtering

of a bucket is below the support threshold all of assigned candidate itemsets of that bucket are deleted from list of candidates.

DHP reduce the database size by not only trimming each individual transaction size but also pruning the number of transaction in database. Note that as observed in [7] on mining association rule, any subset of a large itemset must be a large itemset by itself. This fact suggest that a transaction be used to determine the set of large *(k+1)*-itemset if it consist of *(k+1)* large *k*-itemset in previous pass, otherwise the transaction prune from the database.

We now, take a closer look at how the transaction size is trimmed by DHP. If a transaction contain some *(k+1)* large itemsets, any item contained in these *(k+1)*-itemsets will appear in at least *K* of the candidate *k*-itemsets in $C_{(k)}$. as a result, an item in transaction *T* can be trimmed if does not appear in at least *k* of the candidate *k*-itemsets in *T*. this part is a necessary condition, not a sufficient condition. An example of transaction trimming is shown in Fig. 2.

Our Implementation: In this section, we describe our new algorithm that we used it. In 3.1 we show our distributed hash filtering idea and in 3-2 we explain our distributed transaction trimming. In Fig. 4 a pseudo-code of our algorithm is shown.

Distributed Hash Filtering: Like the lemma about local and global frequent itemsets that presented in [13] we restate the lemma about each bucket in hash table:

Lemma 1: if a bucket of candidate itemsets are not filter in a distributed database, the bucket must not be filtered in at least one local database.

At the end of iterations, in addition to support counts of candidate itemsets, information about hash table of each local database must be exchanged among sites.

In order to better perform of this operation, number of polling sites are determine like the approach which the FDM uses to obtain global support of candidate itemsets. We can assign number of bucket to each polling site. If in a local site the count of a bucket is larger than minimum support threshold, the site sends the bucket to corresponding polling site. The polling site aggregate the information about that bucket from other sites and determines weather that bucket must be filtered or no. Each polling site informs other sites about its results.

In Fig. 4 there is a simple example about Distributed hash Filtering. In this example, there are 2 sites and minimum support threshold is 4 in whole database. Therefore local support in each site is equal to 2. Bucket 0 doesn't be filtered, because count of this bucket in each of two sites is greater than minimum support. Bucket 1 is satisfied by minimum support in site 1 and not in site 2, but sum of this bucket in all of databases is equal to 4 and satisfies minimum support due to keep of this bucket. Values of Bucket 3 in each of two sites are not greater than Minimum support, so we filter this bucket without any communication. But we filter bucket 4 by using its polling site.

Distributed Database Transaction Trimming: In order to improve the performance of FDM algorithm, transaction trimming is also adopted. Each local site used the approach described in section 2.4. In this way the size and the number of transactions of distributed database is reduced. Since each site autonomously and without communicating with other sites performs transaction trimming, this technique can significantly improve the run time of the DARM problem due to distributed database size reduction.

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 3.00GHz 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 [20]. To empirically evaluate the effect of using proposed technique several tests are performed on the datasets kosarak and T40I10D100K. Both dataset are available on FIM repository.

Both the FDM algorithm and a version of the FDM algorithm which uses the new technique are implemented. To implementation of them, we use trie data structure. The aim is to show the effect of using the new technique inside of a DARM algorithm in terms of communication and computation.

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 and 2 experimental results for two 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.

Table 1: Execution times for database Kosarak

minimum support	1 site	2 sites	4 site	6 sites	8 sites
0.04848	17.75	10.93	5.01	3.59	2.41
0.00303	23.33	14.11	6.15	4.95	3.10
0.00202	35.43	20.13	10.3	7.73	5.6
0.00121	159.06	98.33	50.32	38.12	30.31
0.00091	538.64	310.03	169.33	140.12	101.12
0.00088	772	482.68	238.12	190.01	151.12
0.00085	1610.61	1283.3	590.12	483.81	400.78

Table 2: Execution times for database T40I10D100K

minimum support	1 site	2 sites	4 site	6 sites	8 sites
0.03000	10.13	5.78	3.23	2.30	1.63
0.01000	37.23	22.10	11.01	8.05	5.93
0.00800	143.12	88.34	43.92	32.89	25.51
0.00580	325.79	205.33	110.12	84.05	61.95
0.00370	528.04	307.12	165.12	127.40	99.21
0.00215	1535.65	1103.77	510.32	430.91	340.05

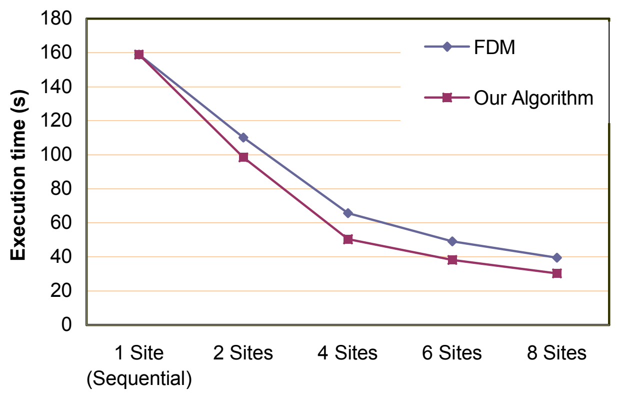


Fig. 5: Execution times for database kosarak with minimum support threshold of 0.00121

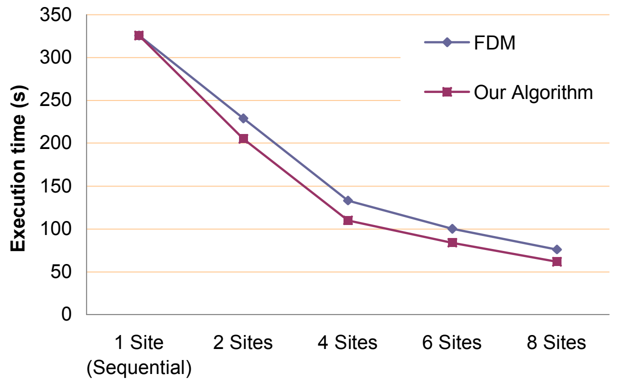


Fig. 6: Execution times for database T40I10D100K with minimum support threshold of 0.00580

In Fig. 5 and Fig. 6, there are two comparisons between our implementation and FDM algorithm in various number of sites. First figure show results on kosarak database by support threshold equals to 0.00121. Second figure illustrate the results by database T40I10D100K and support 0.00580.

CONCLUSION

In this study a new distributed algorithm is proposed to mine all frequent itemsets in distributed environment. This algorithm uses hash filtering to reduce communication and improve the performance and also uses transaction trimming to reduce run time of the distributed data mining task. We use the FDM algorithm as a base to exploit two mentioned technique since the FDM algorithm is a well known algorithm having positive characteristics in message exchange and candidate generation.

Experimental evaluations on real life and syntactical distributed data show the superiority of our new algorithm in terms of communication and computations rather than the FDM and CD algorithms.

REFERENCES

1. Agrawal, R., T. Imielinski and A. Swami, 1993. Mining association rules between sets of items in large databases. In Proc. of the ACM SIG-MOD Conference on Management of Data, pp: 207-216.
2. Jiawei Han, Hong Cheng, Dong Xin and Xifeng Yan, 2007. Frequent pattern mining: current status and future directions, Data Mining and Knowledge Discovery, 15: 55-86.
3. Chen, M.S., J. Han and P.S. Yu, 1996. "Data Mining: An Overview from a Database Perspective, IEEE Transactions on Knowledge and Data Engineering, 8(6): 866-883.
4. Hipp, J., Ulrich Güntzer and Gholamreza Nakhaeizadeh, 2000. Algorithms for association rule mining - a general survey and comparison, ACM SIGKDD Explorations Newsletter, 2(1): 58-64.
5. Agrawal, R., T. Imielinski and A. Swami, 1993. 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.

6. Maurice Houtsma and Arun Swami, 1995. Set-oriented data mining in relational databases, Data and Knowledge Engineering, 17(3): 245-262
7. Agrawal, R. and R. Srikant, 1994. Fast Algorithms for Mining Association Rules, Proceedings of the 20th International Conference on Very Large Data Bases, pp: 487-499.
8. Toivonen, H., T.M. Vijayaraman, A.P. Buchmann, C. Mohan and N.L. Sarda, 1996. Sampling large databases for association rules, In Proceedings 22nd International Conference on Very Large Data Bases, pp: 134-145.
9. Brin, S., R. Motwani, J.D. Ullman and S. Tsur, 1997. 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, pp: 255-264.
10. Han, J., J. Pie, Y. Yin and R. Mao, 2003. Mining frequent pattern without candidate generation: A frequent-pattern tree approach. Data Mining and knowledge discovery.
11. Bodon, F., 2003. "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.
12. Agrawal, R. and J. Shafer, 1996. Parallel mining of association rules. IEEE Transaction on Knowledge and Data Engineering, 8(6): 962-969.
13. Cheung, D.W., *et al.*, 1996. A Fast Distributed Algorithm for Mining Association Rules. In Proc. Parallel and Distributed Information Systems, IEEE CS Press, pp: 31-42.
14. Schuster, A. and R. Wolf, 2001. Communication-Efficient Distributed Mining of Association Rules, In Proc. ACM SIGMOD International Conference on Management of Data, ACM Press, pp: 473-484.
15. Schuster, A., 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.
16. Ashrafi, M.Z., D. Taniar and K. Smith, 2004. ODAM: an Optimized Distributed Association Rule Mining Algorithm, IEEE Distributed Systems Online, 5: 3.
17. Bodon, F., 2004. "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.
18. Bodon, F., 2006. A Survey on Frequent Itemset Mining, Technical Report, Budapest University of Technology and Economic.
19. Park, J.S., M.S. Chen and P.S. Yu, 1997. "Using a hash-based method with transaction trimming for mining association rules", Transactions on Knowledge and Data Engineering, 9(5): 813-825.
20. Snir, M., S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra, 1996. MPI: The Complete Reference, The MIT Press.