Fast Parallel Association Rule Mining Without Candidacy Generation

Osmar R. Zaïane Mohammad El-Hajj Paul Lu
University of Alberta, Edmonton, Alberta, Canada
{zaiane, mohammad, paullu}@cs.ualberta.ca

Abstract

In this paper we introduce a new parallel algorithm MLFPT (Multiple Local Frequent Pattern Tree) [11] for parallel mining of frequent patterns, based on FP-growth mining, that uses only two full I/O scans of the database, eliminating the need for generating the candidate items, and distributing the work fairly among processors. We have devised partitioning strategies at different stages of the mining process to achieve near optimal balancing between processors. We have successfully tested our algorithm on datasets larger than 50 million transactions.

1. Introduction

Association rule mining algorithms currently proposed in the literature are not sufficient for extremely large datasets and new solutions still have to be found. In particular there is a need for algorithms that do not depend on high computation and repeated I/O scans. Parallelization is a viable solution. However, distributing and balancing the mining tasks between the processors without jeopardizing the global solution is not trivial. The problem of mining association rules over market basket analysis was introduced in [1]. Association rules are not limited to market basket analysis, but the analysis of sales, or what is known as basket data, is the typical application often used for illustration. The problem consists of finding associations between items or itemsets in transactional data. The data could be retail sales in the form of customer transactions or even medical images [12]. Association rules have been shown to be useful for other applications such as recommender systems, diagnosis, decision support, telecommunication, etc. This association-mining task can be broken into two steps: A step for finding all frequent k-itemsets known for its associated extreme I/O and a straightforward step for generating confident rules from the frequent itemsets.

1.1 Related Work

Several algorithms have been proposed in the literature to address the problem of mining association rules. One of the key algorithms, which seems to be the most popular in many applications for enumerating frequent itemsets, is the apriori algorithm [3] the foundation of most known algorithms whether sequential or parallel. Park et al. have proposed the Dynamic Hashing and Pruning algorithm (DHP) [9]. However, the trimming and the pruning properties caused some problems that made it impractical in many cases [13]. The partitioning algorithm proposed in [5] reduced the I/O cost dramatically . However, this method has problems in cases of high dimensional itemsets, and it also suffers from the high false positives of frequent items. FP-growth, was recently proposed by Han et al. [8]. This algorithm creates a relatively compact tree-structure that alleviates the multi-scan problem and improves the candidate itemset generation. The algorithm requires only two full I/O scans for the dataset. Our approach presented in this paper is based on this idea. In spite of the significance of the association rule mining and in particular the generation of frequent itemsets, few advances have been made on parallelizing association rule mining algorithms [6, 2]. Most of the work on parallelizing association rules mining on Shared-memory MultiProcessor (SMP) architecture was based on apriori-like algorithms.

Parthasarathy et al. [10] have written an excellent recent survey on parallel association rule mining with shared-memory architecture covering most trends, challenges and approaches adopted for parallel data mining. All approaches spelled out and compared in this extensive survey are apriori-based. These methods not only require repeated scans of the dataset, they also generate extremely large numbers of candidate sets easily approaching 10^{30} candidates in common cases [7].

1.2 Contribution

In this paper, we introduce a new parallel association rules mining algorithm MLFPT, which is based on the FP-growth algorithm [8]. We have implemented this algorithm on a 64 processor SGI 2400 Origin machine, where all experiments were tested using high dimensionality data that are of a factor of hundreds of thousands of items, and transactional sizes that range in tens of gigabytes. A special optimization step is added to achieve better load balancing with the goal of distributing the work fairly among processors for the mining process

2 Multiple Local Parallel Trees

The MLFPT approach we propose consists of two main stages. Stage one is the construction of the parallel frequent pattern trees (one for each processor) and stage two is the actual mining of these data structures, much like the FP-growth algorithm. However, in order to avoid false negatives, where locally infrequent itemsets are pruned inadvertently while they are frequent globally, we need global counters. Though global counters necessitate locking mechanisms for mutual exclusion, that would add significant overhead and waiting time. Our approach with interlinked local counters avoids the need for locking. Thus, we evade the famous ping-pong problem in parallel programs.

2.1 Construction of the Multiple Local Parallel Trees

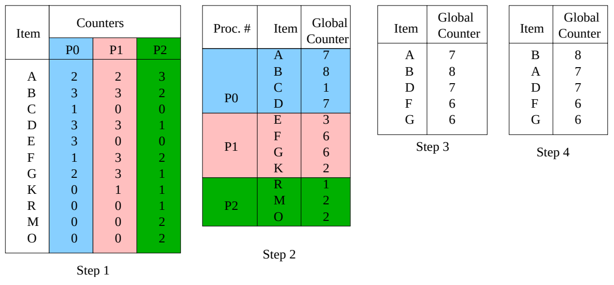
The goal of this stage is to build the compact data structures called Multiple Local Parallel Trees (MLPT). This construction is done in two phases, where each phase requires a full I/O scan for the dataset.

A first initial scan of the database identifies the frequent 1-itemsets. In order to enumerate the frequent items efficiently, we divide the datasets among the available processors. Each processor is given an approximately equal number of transactions to read and analyze. As a result, the dataset is split in p equal sizes. Each processor locally enumerates the items appearing in the transactions at hand. After enumeration of local occurences , a global count is necessary to identify the frequent items. This count is done in parallel where each processor is allocated an equal number of items to sum their local supports into global count. Finally, in a sequential phase infrequent items with a support less than the support threshold are weeded out and the remaining frequent items are sorted by their frequency. This list is organized in a table, called header table, where the items and their respective global support are stored along with pointers to the first occurrence of the item in each frequent pattern tree. Phase 2 would construct a frequent pattern tree for each available processor.

Phase 2 of constructing the MLPT structures is the actual building of the individual local trees. This phase requires a second complete I/O scan from the dataset where each processor reads the same number of transactions as in the first phase. Using these transactions, each processor builds its own frequent pattern tree that starts with a null root. For each transaction read by a processor only the set of frequent items present in the header table is collected and sorted in descending order according to their frequency.

TID	Items Bought	Processor Number
1	A, B, C, D, E	$\rightarrow P_0$
2	F, B, D, E, G	
3	B, D, A, E, G	
4	A, B, F, G, D	$\rightarrow P_1$
5	B, F, D, G, K	
6	A, B, F, G, D	
7	A, R, M, K, O	$\rightarrow P_2$
8	B, F, G, A, D	
9	A, B, F, M, O	

Table 1. Transactional database example.



Item	Counters		
	P0	P1	P2
A	2	2	3
B	3	3	2
C	1	0	0
D	3	3	1
E	3	0	0
F	1	3	2
G	2	3	1
K	0	1	1
R	0	0	1
M	0	0	2
O	0	0	2

Step 1

Proc. #	Item	Global Counter
P0	A	7
	B	8
	C	1
	D	7
P1	E	3
	F	6
	G	6
	K	2
P2	R	1
	M	2
	O	2

Step 2

Item	Global Counter
A	7
B	8
D	7
F	6
G	6

Step 3

Item	Global Counter
B	8
A	7
D	7
F	6
G	6

Step 4

Figure 1. Steps of phase 1.

These sorted transaction items are used in constructing the local FP-Trees as follows: for the first item on the sorted transactional dataset, check if it exists as one of the children of the root. If it exists then increment the support for this node. Otherwise, add a new node for this item as a child for the root node with 1 as support. Then, consider the current item node as the newly temporary root and repeat the same procedure with the next item on the sorted transaction. During the process of adding any new item-node to a given local FP-Tree of a processor p, a link is maintained between this item-node in the tree and its entry in the global header table corresponding to processor p. The header table holds as many pointers per item as there are available processors.

For illustration, we use an example with the transactions shown in Table 1. Let the number of available processors be 3 and the minimum support threshold set to 4. The four steps in phase 1 are shown in Figure 1 and Figure 2 shows the result of the tree building process. For the sake of simplicity, only links from the items A and B are drawn from the header table.

2.2 Mining Parallel Frequent items using MLPT Trees

Building the trees in the first stage is not a final goal but a means with the purpose of uncovering all frequent patterns without resorting to additional scans of the data. The mining process starts with a bottom up traversal of the nodes on the MLPT structures, where each processor mines fairly equal amounts of nodes. The distribution of this traversal work is predefined by a relatively small sequential step that precedes the mining process. This step sums the global sup-

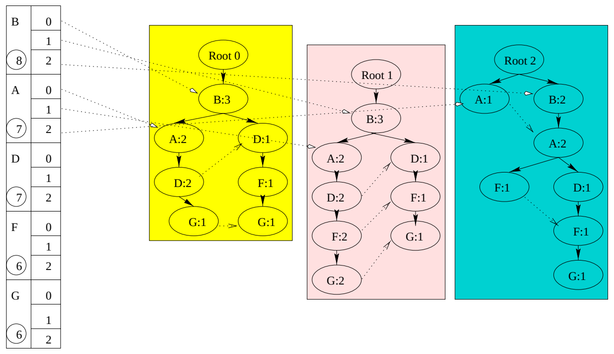


Figure 2. Phase 2 of the construction of the MLPT structure.

Items	Conditional Pattern Base	Conditional FP-Tree
G	(D:1, A:1, B:1) (F:1, D:1, B:1) (F:2, D:2, A:2, B:2) (F:1, D:1, B:1) (F:1, D:1, A:1, B:1)	(B:6, D:6, F:5, A:4)/G
F	(D:1, B:1) (D:2, A:2, B:2) (D:1, B:1) (D:1, A:1, B:1) (A:1, B:1)	(B:6, D:5)/F
D	(A:2, B:2) (B:1) (A:2, B:2) (B:1) (A:1, B:1)	(B:7, A:5)/D
A	(B2) (B:2) (B:2)	(B:6)/A
B	(∅)	

Table 2. Conditional Pattern Bases and the Conditional FPtrees (mining process).

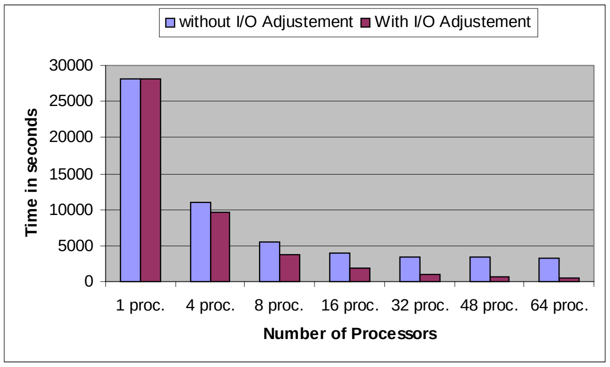


Figure 3. Comparison of execution time for 5 million transactions with and without I/O adjustement.

ports for all items and divides them by the number of processors to find the average number of occurrences that ought to be traversed by each processor. If A is this found average, this sequential step goes over the sorted list of items by their respective support and assigns items consecutively for each processors until the cumulated support is equal or greater than the average A. At this stage all frequent pattern trees are shared by all processors. The task of the processors, once assigned some items, is to generate what is called a conditional pattern base starting from their respective items in the header table. A conditional pattern base is a list of items that occur before a certain item in the frequent pattern tree up to the root of that tree in addition to the minimum support of all the item supports along the list. Since an item cannot only occur in many trees but also in many branches of the same tree, many conditional pattern bases could be generated for the same item. Merging all these conditional pattern bases of the same item yields the frequent string, a string also called conditional FP-Tree, that contains frequent itemsets and their support in the presence of a given item. The merge is based on the items in the patterns and all the supports of the same items are added up in the same manner as in [8]. If the support of an item is less than the minimum support threshold, it is not added in the frequent string.

Table 2 gives all conditional bases and conditional FP-Trees generated from the example in Table 1.

3 Experimental Results

A shared memory SGI Origin 2400 with 64 processors was used to conduct the experiments. We used synthetic transactional databases generated using the IBM Quest synthetic data generator [4]. The sizes of the input databases vary from 1 million transactions to 50 million using dimensions that are multiples of hundreds of thousands. Each of these transactions has at least 12 items preceded by a unique transactional ID. The largest dataset is in the order 10 Gbytes.

In our experiments we studied the MLFPT algorithm with 4, 8, 16, 32, 48 and 64 processors and compared it to its sequential version. The sequential version was, of course, implemented without the summation phase and with only one tree. Speedup measures the performance of parallel execution compared to the sequential execution: $S_p = T_1/T_p$ where S_p is the speedup achieved with p processors, T_1 is the sequential execution time and T_p is the execution time using p processors.

I/O access is normally of an “embarrassingly parallel” nature. For instance, when data is stored on parallel disks with dedicated channels, twice as many processors should read twice as much data. In other words, with appropriate hardware, if it takes t time for one processor to read some data, it should take t/p for p processors to cover the same data.

Since our parallel machine had a sequential disk with one shared head, to assess the real speedup of MLFPT, which does 2 I/O scans of the data regardless of the number of processors, we adjusted the I/O time assuming an "embarrassingly parallel" I/O access.

In our results we decided to adjust the I/O time of our algorithm as follows: The I/O time for parallel execution was estimated using the I/O time for sequential execution divided by the number of processors used. For instance, if using p processors the total execution time is T and the isolated I/O time is t, the execution time with I/O adjusted is calculated $T\prime = T - t + (S/p)$, where S is the isolated I/O time for a sequential execution. In other words, we replaced the 2 scans I/O time recorded with the expected real parallel I/O time.

Due to the space limitation we will only present figure 3 that depicts the significant time reduction with the increase of processors when mining 5 million transactions.

MLFPT operations are divided into two stages where most of the computation in the MLFPT algorithm is done during building the MLPT trees, and then mining them. Building the frequent pattern trees, which utilize most of the processing time, is shown to be of "embarrassingly parallel" nature and this indeed was the reason for the several-fold improvements achieved as we increased the number of processors in our experiments. This is due to the fact that the work is evenly partitioned among the processors and each unit of work is completely independent of each other where each processor builds a sub-tree representing its partition of transactions. There is no ping-pong effect where processors are waiting for each other.

Our experiments have shown that this creation and mining is almost linearly proportional to the number of processors and the size of the transactional datasets, where the speedup of the MLFPT algorithm increases as the problem size increases. These results suggest that the MLFPT algorithm would achieve speedups for extremely large datasets as well.

4 Conclusion and Future Work

In this paper, we have introduced an efficient parallel implementation of an FP-Tree-based association rule mining algorithm and have proposed a solution for load balancing among processors and resource sharing with minimum mutual-exclusion locking. We have discussed our experiments with this new parallel algorithm, MLFPT, for mining frequent patterns without candidate generation. The MLFPT algorithm overcomes the major drawbacks of parallel association rule mining algorithms derived from apriori, in particular the need for k I/O passes over the data.

Our experiments showed that with I/O adjusted, the MLFPT algorithm could achieve an encouraging many-fold speedup improvement.

The implementation of our algorithm and the experiments conducted were on a shared memory and shared hard drive architecture. We have recently acquired a cluster with 8 dual processor nodes and we plan to investigate the same approach with shared nothing architecture and devise a new protocol for sharing global resources while minimizing the message passing overhead. We are in the process of experimenting our algorithms with up to 1 billion transactions.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.

[2] R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation, and experience. *IEEE Trans. Knowledge and Data Engineering*, 8:962–969, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[4] I. Almaden. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/quest/syndata.html.

[5] S. Brin, R. Motwani, J. D. Ullman, and S.Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 255–264, Tucson, Arizona, May 1997.

[6] D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. 1996 Int. Conf. Parallel and Distributed Information Systems*, pages 31–44, Miami Beach, Florida, Dec. 1996.

[7] J. Han and M. Kamber. *Data Mining, Concepts and Techniques*. Morgan Kaufmann, 2001.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM-SIGMOD*, Dallas, 2000.

[9] J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, pages 175–186, San Jose, CA, May 1995.

[10] S. Parthasarathy, M. J. Zaki, and M. Ogihara. Parallel data mining for association rules on shared-memory systems. *Knowledge and Information Systems: An International Journal*, 3(1):1–29, February 2001.

[11] O. R. Zaïane, M. El-Hajj, and P. Lu. Fast parallel association rule mining without candidacy generation. Technical Report TR01-12, Department of Computing Science, University of Alberta, Canada, August 2001. ftp://ftp.cs.ualberta.ca/pub/TechReports/2001/TR01-12/TR01-12.pdf.

[12] O. R. Zaïane, J. Han, and H. Zhu. Mining recurrent items in multimedia with progressive resolution refinement. In *Int. Conf. on Data Engineering (ICDE'2000)*, pages 461–470, San Diego, CA, February 2000.

[13] M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency, Special Issue on Parallel Mechanisms for Data Mining*, 7(4):14–25, December 1999.