Parallel Algorithm for Mining Maximal Frequent Patterns[*]

Hui Wang[1,2**], Zhiting Xiao[2], Hongjun Zhang[1], and Shengyi Jiang[1]

[1] Computer School,
Huazhong University of Science & Technology,
430074, WuHan, China
[2] Wuhan Communication College,
430010,Wuhan, China
hustwanghui@mail.china.com

Abstract. We present a novel and powerful parallel algorithm for mining maximal frequent patterns, called Par-MinMax. It decomposes the search space by prefix-based equivalence classes, distributes work among the processors and selectively duplicates databases in such a way that each processor can compute the maximal frequent patterns independently. It utilizes multiple level backtrack pruning strategy and other novel pruning strategies, along with vertical database format, counting frequency by simple tid-list intersection operation. These techniques eliminate the need for synchronization, drastically cutting down the I/O overhead. The analysis and experimental results demonstrate the superb efficiency of our approach in comparison with the existing work.

1 Introduction

Mining frequent patterns is to discover all frequent patterns in a given database. It comprises the core of several data mining algorithms such as association rule mining and sequence mining, and dominates the running time of those algorithms. It has been shown to have an exponential worst case running time in the number of items, therefore much research [1,2,3,4] has been devoted to increasing the efficiency of the task.

Since both the data size and the computational costs are large, parallel algorithms have been studied extensively. Frequent pattern discovery has become a challenge for parallel programming since it is a highly complex operation on huge datasets demanding efficient and scalable algorithms. Most previous parallel algorithms [5,6,7,8,9,10] use complicated hash structures, make repeated passes over the database partition, have to exchange the partial results among all the processors during each iteration, resulting in additional maintaining overhead, high I/O overhead, expensive communication and synchronization cost.

[*] This paper is supported by the National Natural Science Foundation of China under Grant No.60273075.

[**] Hui Wang is a PhD candidate in computer school, Huazhong university of science and technology. Her research interests include data mining and parallel computing.

We present a novel and powerful parallel algorithm for mining maximal frequent patterns, called Par-MinMax, which is based on its serial version MinMax[11]. The new algorithm decomposes the original search space into smaller pieces by prefix-based equivalence classes, schedules classes among processors by the weights, distributes work among the processors and selectively duplicates databases in such a way that each processor can compute the frequent patterns independently. It uses depth-first search and a novel multiple level backtrack pruning strategy[11] and other powerful pruning strategies, along with vertical tid-list database format, counting frequency by simple tid-list intersection operation. These techniques eliminate the need for synchronization, drastically cutting down the I/O overhead. The analysis and experimental results demonstrate the superb efficiency of our approach in comparison with the previous work.

The rest of this paper is organized as follows: In section 2 we describe the maximal frequent pattern problem. The serial version MinMax is briefly described in Section 3. Section 4 describes our new algorithm Par-MinMax. We show the experimental results in section 5. The conclusions are in section 6.

2 Problem Statement

The problem of mining maximal frequent patterns is formally stated by definitions 1-4 and theorems 1-2. To describe our algorithm clearly, definition 5-9, propositions 1-2 and theorem 3 are given in this paper.

Let $\mathbf{I}=\{i_1,i_2,......,i_m\}$ be a set of m distinct items. Let **D** denote a database of transactions where each transaction has a unique identifier (tid) and contains a set of items.

Definition 1: (pattern)
A set $X \subseteq \mathbf{I}$ is called a pattern (an itemset). A pattern with k items is called a k-pattern.

Definition 2: (pattern's frequency σ)

The frequency of a pattern X, denoted by σ (X), is defined as the number of transactions in which the pattern occurs as a subset, called the support of the pattern.

With the vertical database layout, the database is comprised by items with corresponding tid-lists.

Let $x \in \mathbf{I}$, tid-list(x)=$\{t_i | x$ appeared in $t_i\}$. $\sigma(x)$=|tid-list(x)|. Let $X \subseteq \mathbf{I}$, $X=\{x_1,x_2,......,x_k\}$, $\sigma(X)=|\bigcap_{xi \in X} tid-list(xi)|$.

Definition 3: (frequent pattern)

Let ε be the threshold minimum frequency value specified by user. If $\sigma(X) \geq \varepsilon$, X is called a frequent pattern. The frequent 1-pattern is called a frequent item. The set of all the frequent k-patterns in **D** is denoted by F_k.

Definition 4: (maximal frequent pattern)
If $\sigma(X) \geq \varepsilon \wedge \neg\exists Y(Y \supset X) \wedge \sigma(Y) \geq \varepsilon$, we say X is a maximal frequent pattern.

Let $\mathbf{F}=\{X \mid X \subseteq \mathbf{I} \wedge \sigma(X) \geq \varepsilon\}$, $\mathbf{M}=\{X \mid X \in \mathbf{F} \wedge \neg\exists Y (Y \supset X) \wedge Y \in \mathbf{F}\}$, so $\mathbf{M} \subseteq \mathbf{F}$. Since a maximal frequent m-pattern includes 2^m frequent patterns, $|\mathbf{M}| << |\mathbf{F}|$. Given a threshold minimum frequency value ε and a database **D**, the mining goal is to find **F** from **D**. Since the maximal frequent patterns contain all the frequent ones, it is wise to compute **M** instead of **F**.

It is profitable to view the frequency mining as a search problem. Each node in search space is composed of two parts: *head* and *tail*. In the initial status, the only node is root where *head*=$\varnothing$ and *tail*=F_1. The number of items in *node*'s *tail* is the number of nodes in the next level that the node *node* can be frequently extended. Let *node*'s *tail*=$\{a_1,a_2,\ldots\ldots,a_n\}$, where *head*$\bigcup\{a_i\}$ is frequent, i=1, 2, …, n, then node (*head*, *tail*) can be extended n nodes: *head*$\bigcup\{a_i\}$, $\{a_{i+1}, a_{i+2}, \ldots\ldots, a_n\}$, i=1,2,……,n. The goal is to find all the *head*s. We use head$\bigcup$tail to represent the node itself.

According to definition 3, the following theorems hold.

Theorem 1: Any sub_patterns of a frequent pattern are frequent.

Theorem 2: Any super_patterns of an infrequent pattern are infrequent.

Definition 5: (item's infrequency λ)
A frequent item x's infrequency λ is defined as the number of infrequent 2-patterns it makes.

$$\lambda(x) = |\{y \mid y\in F_1 \wedge x\in F_1 \wedge \sigma(\{x,y\}) < \varepsilon\}|$$

Proposition 1: If $\lambda(x_1) > \lambda(x_2)$, then x_2 makes more infrequent patterns than x_1.

Proof: Due to the definition 5, the proposition holds.

Proposition 2: If $\sigma(x_1) < \sigma(x_2)$, then x_1 makes more infrequent patterns than x_2.

Proof: Let T be a transaction in **D**, x_1, x_2, $y\in$ **I**. Let P(x) be the probability that x occurs in T. Let P(xy) denote the probability that both x and y occur in T. Since $\sigma(x_1) < \sigma(x_2)$, so $P(x_1)<P(x_2)$. Because $P(xy)=P(x)*P(y)$, so $P(x_1y)<P(x_2y)$, then $\sigma(x_1y) < \sigma(x_2y)$. This implies that x_1 makes more infrequent patterns than x_2.

Theorem 3: Let P be maximal frequent pattern in the sub-tree rooted with node *node*(*head*,*tail*). If P=*head*$\bigcup$*tail*, then all nodes in this sub-tree are frequent.

Proof: In terms of the extending process, for any node *son-node* in the sub-tree rooted with *node*, *son-node*$\subseteq$ *head*$\bigcup$*tail* holds. Since P=*head*$\bigcup$*tail*, we have *son-node*$\subseteq$P. As P is frequent, so *son-node* is frequent.

Definition 6: (equivalence relation)
Let P be a set. An equivalence relation on P is a binary relation $\equiv$ such that for all X, Y, Z $\in$ P, the relation is:
1) Reflexive: $X \equiv X$.
2) Symmetric: $X \equiv Y$ implies $Y \equiv X$.
3) Transitive: $X \equiv Y$ and $Y \equiv Z$, implies $X \equiv Z$.

Definition 7: (equivalence class)

The equivalence relation partitions the set P into disjoint subsets called equivalence classes. The equivalence class of an element X $\in$ P is given as [X]={Y|Y$\in$ P$\wedge$X$\equiv$Y}.

Define a function p: P(**I**)$\times$N$\mapsto$P(**I**) where p(X, k) = X[1:k], the k length prefix of X.

Definition 8: (prefix-based equivalence relation)
Define an equivalence relation θ_k on the lattice P(**I**) as follows: $\forall$ X,Y$\in$ P(**I**), X θ_k Y $\Leftrightarrow$ p(X, k) = p(Y, k). That is, two patterns are in the same class if they share a common k length prefix. We therefore call θ_k a prefix-based equivalence relation.

Definition 9: (equivalence class weight)
Let [x] denote an equivalence class on F_2, based on equivalence relation θ_1, *m* be the number of class [x], ω(x) denote the weight of class [x], ω(x)= $\sum_{y:\{x\}\cup\{y\}\in[x]} \lambda(y)$ */m.*

3 MinMax: An Efficient Serial Algorithm

MinMax is an iterative algorithm based on a depth-first traversal over the search tree rooted with F_1 and returns the exact set of M. The basic idea of MinMax is to find out maximal frequent patterns as soon as possible and to use them to prune away the non-maximal frequent patterns which have superset in M. It has a stack keeping search trace and performs multiple level backtrack pruning (see line 7). Initially, F_1 was sorted by $\lambda\downarrow\sigma\uparrow$ according to propositions 1 and 2. It makes head with smaller tail and gets to a maximal frequent pattern much faster. MinMax also uses pruning strategies based on theorems 1(see line 6) and 2(see line 5) to make the process more efficiently.

```
MinMax(F1,M)/* MinMax outputs the set of all the maximal frequent patterns M, F1 was the input */
1. sort $F_1$ by $\lambda\downarrow\sigma\uparrow$;
2. stack P was initialized as ($\varnothing$,$F_1$,0);
3. select the most left item $a_i$ in tail which flagbits($a_i$) is 0;
4. current_head $\leftarrow$ P.head $\cup$ {$a_i$};
5. current_tail:={y|y$\in$ P.tail $\wedge$ y>$a_i$ $\wedge$ current_head $\cup$ {y} is frequent};
6. if current_head $\cup$ current_tail has a superset in M then flagbits($a_i$) $\leftarrow$ 1; goto 3
   else goto 7
7. if current_tail==$\varnothing$ and current_head has no superset in M then
   M=M $\cup$ current_head; backtrack to the oldest ancestor which
   head $\cup$ tail==current_head; flagbits($a_i$) $\leftarrow$ 1; goto 3
8. if current_tail$\neq\varnothing$ then push (current_head, current_tail, 0); goto 3
```

```
9.  if all the flagbits==1 then x0 ⟵ the last item in P.head; pop; flagbits(x0) ⟵ 1;
    goto 3
10. if the stack P's end status is arrived then return M.
End
```

4 Par-MinMax: Algorithm Design and Implementation

The new algorithm Par-MinMax overcomes the shortcomings of the Count and Candidate Distribution algorithms. It utilizes the aggregate memory of the system by partitioning the patterns into disjoint sets, which are assigned to different processors. The dependence among the processors is decoupled right in the beginning. Since each processor can proceed independently, there is no costly synchronization. Furthermore the new algorithm uses the vertical database layout which clusters all relevant information in an pattern's tid-list. Each processor computes all the frequent patterns from one class before proceeding to the next. The local database partition is scanned only once. As the pattern size increases, the size of the tid-list decreases, resulting in very fast intersections.

There are three distinct phases in the algorithms. The initialization phase, responsible for scheduling equivalence classes and distributing related tid-lists among the processors; the asynchronous phase, which generates local maximal frequent patterns, and the final reduction phase, which kicks out all the local maximal but not maximal frequent patterns in global. The more detail is as following:

(1) Equivalence Class Generating and Scheduling: We first partition F_2 into equivalence classes by θ_1, then computer each class's weight by definition 9, and sort the classes on the weights. We use a greedy algorithm to schedule the classes among the processors by assigning each class in turn to the least loaded processor at that point.

(2) Database Repartitioning: Database was partitioned roughly equal among processors. To minimize communication and make each processor work independently, each processor scans the item tid-lists in its local database partition and writes it to a transmit region which is mapped for receive on other processors. The other processors extract the tid-list from the receive region if it belongs to any class assigned to them.

(3) Asynchronous Mining: At the end of the initialization step, the relevant tid-lists are available locally on each host, thus each processor can independently generate the maximal frequent patterns from its assigned classes eliminating the need for synchronization with other processors. Each class is processed in its entirety before moving on to the next class in the schedule.

(4) Final Post-processing: Since the results by each processor might be local maximal but not global maximal frequent patterns, so the non-maximal frequent patterns will be killed in the final post-processing.

The new algorithm Par-MinMax is described as follows:

```
Par-MinMax($F_1$,M)/* Par-MinMax outputs all the maximal frequent patterns, $F_1$ is the input */
/* Initialization phase */
Generate independent classes from $F_2$ by $\theta_1$;
Schedule the classes among the processors on the weight of each class, by definition 9.
Scan local database partition ; Transmit relevant tid-lists to other processors; Receive tid-lists from other processors;
/* Asynchronous Phase */
for each processor $P_i$:
for each assigned class [x]:
$Y=\{y|\{x\} \cup \{y\} \in [x]\}$;
MinMax(Y,$M_i$);
/* Final post-process Phase */
Aggregate Results and kick out all the non-maximal patterns and Output M
End
```

Table 1. The synthetic databases

Database	T	I	D	D1	D2	D4	D6
T10.I4.D2084K	10	4	2,084,000	91MB	182MB	364MB	546MB
T15.I4.D1471K	15	4	1,471,000	93MB	186MB	372MB	558MB
T20.I6.D1137K	20	6	1,137,000	92MB	184MB	368MB	552MB

Table 2. Speedup experiments

Number of processors	T10.I4.D2084 K	T15.I4.D1471K	T20.I6.D1137K
1	144	315	810
2	81	225	495
4	72	180	360
8	54	135	262

Table 3. Sizeup experiments

r	T10.I4.D2084K	T15.I4.D1471K	T20.I6.D1137K
1	20	40	80
2	35	60	150
4	72	180	360
6	158	320	700

5 Experimental Results

We implemented the algorithms on Dawn 3000 with 3 hosts, each host has 4 processors shared 2GB RAM, 9GB hard disk, each processor has the CPU 375MHz, by several synthetic datasets. It shows that our parallel algorithm Par-MinMax is well scalable in speedup and in sizeup.

We use the synthetic databases shown in table 1 to test our algorithm. Where T denotes the average transaction size , I the average maximal potentially frequent pattern size, D_r the number of transactions , where r is the replication factor. For r = 1, all the databases are roughly 90MB in size. In speedup experiments, r=4. In sizeup experiments, the number of processors is 4. The speedup and sizeup experiment results are shown in table 2 and table 3 respectively. It shows that the speedup and sizeup are nearly linear.

6 Conclusions

In this paper we proposed a new parallel algorithm Par-MinMax for mining maximal frequent patterns. The algorithm uses the prefix-based decomposition technique, and the multiple level backtrack pruning strategy. The set of independent classes is scheduled among the processors, and the database is also selectively replicated so that the portion of the database needed for the computation of frequency is local to each processor. After the initial setup phase the algorithm does not need any further communication or synchronization. We implemented the algorithms on Dawn 3000 by several synthetic datasets. It shows that our parallel algorithm Par-MinMax is well scalable in speedup and in sizeup. Naturally, the load balance schema used by most parallel mining algorithms are static, our current research is directed to applying dynamic load balance schema to our algorithm, it would be hopeful to achieve better performance.

References

1. R.Agrawal, H.Mannila, R.Srikant, H.Toivonen and A.I. Verkamo: Fast Discovery of Association Rules. Advances in Knowledge Discovery and Data Mining, Chapter 12, AAAI/MIT Press, 1995.
2. R. J. Bayardo Jr.. Efficiently Mining Long Patterns from Databases. *Proc. of the ACM SIGMOD Conference on Management of Data,* Seattle, *pages 85-93*, June 1998.
3. D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: a maximal frequent itemset algorithm for transactional databases. In Intl. Conf. on Data Engineering, Apr. 2001.
4. K. Gouda, M. J. Zaki. Efficiently Mining Maximal Frequent Itemsets. In 1st IEEE Intl. Conf. On data mining, Nov. 2001.
5. R. Agrawal and J. Shafer, Parallel Mining of Association Rules, IEEE Trans. on Knowledge and Data Engg., 8(6):962–969, December 1996.
6. M.J.Zaki, S. Parthasarathy, M. Ogihara, and W. Li, New parallel algorithms for fast discovery of association rules, Data Mining and Knowledge Discovery: An International Journal, 1(4):343–373, December 1997.

7. M.J.Zaki: Parallel and Distributed Association Mining: A Survey, IEEE Concurrency, Vol.7, No.4, pp.14–25, 1999.
8. Ruoming Jin and Gagan Agrawal. Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance. In Proceedings of the second SIAM conference on Data Mining, April 2002.
9. Srinivasan Parthasarathy, Mohammed Zaki, Mitsunori Ogihara, Wei Li, Parallel Data Mining for Association Rules on Shared-memory Systems, in Knowledge and Information Systems, Volume 3, Number 1, pp 1–29, Feb 2001.
10. Karlton Sequeira, Mohammed J. Zaki, ADMIT: Anomaly-base Data Mining for Intrusions, 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002.
11. H.Wang, Q.Li, A Maximal Frequent Itemset Algorithm, Lecture Notes in Computer Science 2639, May 2003.