Efficient Parallel Mining of Association Rules on Shared-Memory Multiple-Processor Machine

Kan Hu† David W. Cheung‡ Shaowei Xia†

† Department of Automation, Tsinghua University, Beijing 100084, China.
Email: hukan@cs.hku.hk, swxia@mail.tsinghua.edu.cn.
‡ Department of Computer Science, The University of Hong Kong, Hong Kong.
Email: dcheung@cs.hku.hk.

China

Abstract **– In this paper we consider the problem of parallel mining of association rules on a shared-memory multiprocessor system. Two efficient algorithms PSM and HSM have been proposed. PSM adopted two powerful candidate set pruning techniques *distributed pruning* and *global pruning* to reduce the size of candidates. HSM further utilized an I/O reduction strategy to enhance its performance. We have implemented PSM and HSM on a SGI Power Challenge parallel machine. The performance studies show that PSM and HSM out perform CD-SM, which is a shared-memory parallel version of the popular Apriori algorithm.**

I. INTRODUCTION

Mining *association rules* in large databases has attracted a lot of attention in data mining research [1, 2, 4, 8]. The mining process needs to scan all the transactions in the database, which introduces a significant amount of I/Os. In addition, it has to search through a large number of candidates for large itemsets which demands a lot of CPU computation. Therefore, the development of parallel algorithms for mining association rules is an important problem. In this work, we attempt to solve this problem on shared-memory multiple-processor machines such as the SGI Power Challenge.

Most proposals in parallel mining have been focused on distributed or shared-nothing model [5, 6, 7, 9, 10]. In those models, the database is partitioned and distributed in the local disk of the processors. The memory limitation and I/O cost are the dominating performance factors. The shared-memory multiprocessor parallel machine is another important computing model. But very few parallel mining works have been carried out on this model. A direct extension of the Apriori algorithm to the shared-memory model has been presented in [11]. Some important characteristics of the candidate sets in a partitioned database have been discovered [5]. They have been used to design two effective two pruning techniques, the *distributed pruning* and *global pruning*, which can reduce the amount of candidates effectively in the distributed or parallel environment.

One of the first parallel algorithm in shared-nothing model is the CD (Count Distribution) algorithm [3]. For comparison purpose, we have realized the CD algorithm on a SGI Power Challenge shared-memory multiprocessor machine. This version of CD is called CD-SM (CD on Shared-memory Model).

In this work, we propose two efficient algorithms on the shared-memory multi-processor machine. The first algorithm is PSM (Parallel mining on Shared-memory Model) which adopts the two pruning techniques to reduce the number of candidates in each iteration. PSM remedies the problem of large number of candidate sets in the CD-SM algorithm. So PSM consumes less computing time than the CD-SM. However, PSM still needs to perform the same number of scannings on the database. In general, the number of candidates after iteration two would reduce drastically. Therefore, we have reduced the number of scannings by combining the computations of all the iterations after iteration two. This enhanced version is the HSM (Hybrid parallel mining on Shared-memory Model) algorithm. Therefore, HSM can reduce the I/O cost at most situations (i.e., when need to mine more than two iterations in CD-SM), and has less CPU cost compared with CD-SM.

We have implemented the above algorithms on a SGI Power Challenge shared-memory muti-processor machine with 8 processors. All algorithms base on the framework of common candidate partitioned database. One single candidate hash tree or Trie is used by all the processors, while the database is partitioned among them. Each processor traverses its local database and stores the support for itemsets separately on the shared hash tree. Finally, a master process computes the large itemsets according to the given threshold. Extensive performance studies have been carried out. It was observed that both PSM and HSM performed faster than CD-SM. In particular, HSM enjoys a very good reponse time due to its I/O reduction. The performance studies also showed that PSM and HSM have better speed up property than CD-SM.

0-7803-4253-4/97/$10.00 © 1997 IEEE

The rest of this paper is organized as follows. Section 2 overviews the parallel mining of association rules. Two candidate pruning techniques, *distributed pruning* and *global pruning* are described in Section 3. In Section 4, we present the PSM and HSM algorithms. Section 5 reports the result of the performance study. Finally we conclude in Section 6.

II. PARALLEL MINING OF ASSOCIATION RULES

A. Association Rules

Let $I = \{i_1, i_2, \cdots, i_m\}$ be a set of items and D be a database of transactions, where each transaction T consists of a set of items such that $T \subseteq I$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \phi$. An association rule $X \Rightarrow Y$ has support s in D if the probability of a transaction in D contains both X and Y is s. The association rule $X \Rightarrow Y$ holds in D with confidence c if the probability of a transaction in D which contains X also contains Y is c. The task of mining association rules is to find all the association rules whose support is larger than a given minimum support threshold and whose confidence is larger than a given minimum confidence threshold. For an itemset X, we use $X_{.sup}$ to denote its *support count* in database D, which is the number of transactions in D containing X. An itemset $X \subseteq I$ is *large* (or *frequent*) if $X_{.sup} \geq minsup \times |D|$, where $minsup$ is the given minimum support threshold. For the purpose of presentation, we sometimes just use *support* to stand for support count of an itemset.

It has been shown that the problem of mining association rules can be decomposed into two subproblems [1] : (1) find all large itemsets for a given minimum support threshold, and (2) generate the association rules from the large itemsets found. Since (1) dominates the overall cost, the current research has been focused on how to efficiently solve the first subproblem.

B. Count Distribution Algorithm for Parallel Mining

Apriori is the most well known serial algorithm for mining association rules [2]. It relies on the *apriori_gen* function to generate the candidate sets at each iteration. CD (Count Distribution) is a parallel version of *Apriori* for parallel mining basing on shared-nothing multiprocessor [3]. The database D is partitioned into $D_1, D_2, \cdots, D_n$ and distributed across n processors. The program fragment of CD at processor i, $1 \leq i \leq n$, for the k-th iteration is outlined in Fig. 1. In step 1, every processor computes the same candidate set C_k by applying the *aprior_gen* function on L_{k-1}, which is the set of large itemsets found at the $(k-1)$-th iteration. In step 2, local support counts of candidates in C_k are found. In steps 3 & 4, local support counts are exchanged with all other processors to get global support counts and globally large itemsets L_k are computed independently by each processor. CD repeats steps 1 – 4 until no more candidate is found.

1) C_k = apriori_gen(L_{k-1});
2) scan partition D_i to find the local support count $X_{.sup(i)}$ for all $X \in C_k$;
3) exchange $\{X_{.sup(i)} \mid X \in C_k\}$ with all other processors to get global support counts $X_{.sup}$, for all $X \in C_k$;
4) $L_k = \{X \in C_k \mid X_{.sup} \geq minsup \times |D|\}$

Fig. 1. Count Distribution Algorithm

III. CANDIDATE PRUNING TECHNIQUES

Suppose the entire database D is partitioned into D_1, D_2, $\cdots$, D_n and distributed over n processors. Let X be an itemset and $X_{.sup}$ be the support of X in D. We call $X_{.sup}$ the *global support* of X. Also, we use $X_{.sup(i)}$ to denote the *local support* of X at processor i, which is the support of X in D_i. X is *globally large* if $X_{.sup} \geq minsup \times |D|$. Similarly X is *locally large* at processor i if $X_{.sup(i)} \geq minsup \times |D_i|$. We also call X *gl-large* at processor i, if X is globally large and locally large at processor i. For convenience, we use the term k-itemset to stand for size-k itemset, and use L_k, $GL_{k(i)}$ to denote the set of all globally large k-itemsets and the set of all gl-large k-itemsets at processor i, respectively.

CD only applies function *apriori_gen* on the set L_{k-1} to generate the candidate sets C_k in the k-th iteration. In fact, after the support counts exchange in the $(k-1)$-th iteration, each processor can find out not only the large itemsets L_{k-1} in C_{k-1} but also the processors at which an itemset X is gl-large for any $X \in L_{k-1}$. By using this information, many candidates in C_k can be identified to be small and hence pruned away before the next scan of the database.

A. Distributed Pruning

The *distributed pruning* technique is derived from the observation that all subsets of any large itemsets must be *gl-large* simultaneously on at least one processor. For example, suppose the database is partitioned into D_1 and D_2 on processors 1 and 2. Further assume that both A and B are two size-1 globally large itemsets. In addition, A is gl-large at processor 1 but not processor 2, and B is gl-large at processor 2 but not processor 1. It can be shown that $AB \in C_2$ can never be globally large. If AB is globally large, it must be globally and locally large (gl-large) at some processor. Assume it is gl-large at processor 1, then B must also be gl-large at processor 1, which is contradictory to the assumption. Similarly, AB cannot be gl-large at processor 2. Hence AB cannot be globally large at all. In other words, if AB was globally large,

then A and B must be *gl-large* at the same time on processor 1 or processor 2 or both of them. This observation can be generalized to the k-th iteration. Therefore, the candidates can be generated by applying function *apriori_gen* on each $GL_{k-1(i)}$, $(1 \leq i \leq n)$, independently. The set of size-k candidates generated with this technique is equal to $CG_k = \cup_{i=1}^{n} CG_{k(i)}$, where $CG_{k(i)} =$ apriori_gen$(GL_{k-1(i)})$. Note that the function *apriori_gen* is the same as that in the Apriori algorithm, but it is applied on subsets of L_{k-1} rather than the whole L_{k-1}. Due to the combinatorial effect, the size of $C_k =$ apriori_gen(L_{k-1}) could be much larger than that of CG_k. The above observation can be summarized by the following theorem proved in [5].

Theorem 1 *For $k > 1$, the set of all globally large k-itemsets L_k is a subset of $CG_k = \cup_{i=1}^{n} CG_{k(i)}$, where $CG_{k(i)} = apriori\_gen(GL_{k-1(i)})$.*

Based on Theorem 1, we can prune away any size-k candidate such that there does not exist any processor at which all its size-$(k-1)$ subsets are gl-large. This pruning technique is called *distributed pruning.*

B. Global Pruning

In the counting process, each processor keeps its local support counts for every candidates. The local support counts are exchanged or shared after each iteration. As a result, the local support counts $X_{.sup(i)}$, for all processor i, $(1 \leq i \leq n)$, are also available at every other processor. With this information, another powerful pruning technique called *global pruning* can be developed.

Let X be a candidate k-itemset. At each processor i, $X_{.sup(i)} \leq Y_{.sup(i)}$, where $Y \subset X$. Therefore $X_{.sup(i)}$ is bounded by the value $\min\{Y_{.sup(i)} \mid Y \subset X, \text{ and } |Y| = k-1\}$. Hence the value

$$X_{.maxsup} = \sum_{i=1}^{n} X_{.maxsup(i)}$$

where $X_{.maxsup(i)} = \min\{Y_{.sup(i)} \mid Y \subset X,\ |Y| = k-1\}$

is an upper bound of the global support of X. If $X_{.maxup} <$ minsup$\times|D|$, then X can be pruned away. This technique is called *global pruning.* Note that global pruning requires no additional information except the local support counts resulted from count exchange or sharing in the previous iteration. We can apply global pruning to the survivals of candidates after going through the distributed pruning to get the smaller candidate itemsets. That is, if the upper bound of an itemset X is found to be smaller than the support threshold, X cannot be globally large and should be removed from the set of candidate sets.

IV. PSM AND HSM ALGORITHMS

A. Parallel Mining on Shared-Memory Model Algorithm (PSM)

The PSM is an enhancement of CD-SM. The main difference between them is that both the *distributed pruning* and *global pruning* are incorporated in the PSM algorithm to reduce the candidate set size.

The first iteration of PSM is the same as CD-SM. Each processor scans its partition to find out local support counts of all size-1 itemsets and the master process is in charge of computing the global support counts. At the end, in addition to L_1, each processor also find out the gl-large itemsets $GL_{1(i)}$, for $1 \leq i \leq n$.

For the k-th iteration of FPM, $k > 1$, the program fragment at processor i, $1 \leq i \leq n$, is described in Fig. 2.

1) compute candidate sets $CG_{k(i)} = Apriori\_gen(GL_{k-1(i)})$; (distributed pruning)
2) prune candidates in $CG_{k(i)}$ by global pruning;
3) build $CG_{k(i)}$ into the common hash tree $HT_{(k)}$;
4) scan partition D_i to find the local support count $X_{.sup(i)}$ for any $X \in HT_{(k)}$;
5) compute $GL_{k(i)} = \{X \in HT_{(k)} \mid X_{.sup} \geq minsup \times |D|, X_{.sup(i)} \geq minsup \times |D_i|\}$, for all i, $1 \leq i \leq n$;
6) return $L_k = \cup_{i=1}^{n} GL_{k(i)}$.

Fig. 2. The PSM Algorithm

The PSM algorithm is designed on the model of common candidate hash tree and partitioned database. Every processor can visit the shared hash tree for looking up the candidates while the database split among them. Local counter array with length of $|C_k|$ is kept by every processor to record the local support counts. At the end of each iteration, these local counter arrays are shared by all processors. The data structure of the common hash is exactly same as used in *Apriori* [2].

B. Hybrid Parallel Mining on Shared-Memory Model Algorithm(HSM)

Although PSM has much less candidates than CD-SM, they have the same I/O cost, i.e., they scan database with the same number of passes. Under the shared-memory computing model, most machines only have serial I/O ability up to now. However, we have to face the hugh volume database in the data mining task. Thus the cost of I/O becomes the bottleneck.

The HSM algorithm is designed to reduce the I/O cost. HSM retains the pruning techniques in PSM. In details, the first and second iterations of HSM is the same as PSM. Then we can get the results of L_1, L_2. Obviously C_3 can be generated by using *Apriori_gen* and pruning techniques on L_2. The subsequent steps are different from that in PSM. We continue to generate C_4 from C_3 by

using *Apriori_gen* only and from C_4 to C_5 and so on, until no candidate is generated. A Trie as described in [4] is used to store the support counts after each processor performs one scan on its partition. Therefore, with only one pass, we can compute all large itemsets of size larger than two.

Since HSM only scan database at most three passes, it incures much less I/O comparing with CD-SM. The other more flexible strategy in HSM is to assign a *threshold* for the C_k. When the size of C_k is less than the *threshold*, the algorithm then switchs from the hash tree approach to the Trie approach.

V. PERFORMANCE EVALUATION

All the experiments were performed on a 8-node SGI Power Challenge shared-memory multiprocessor. Each node is a MIPS R10000 processor. There is a total of 512MB of main memory. All processors run IRIX 6.2.

A. Synthetic Databases Generation

We use the synthetic test data generator introduced in [2]. The database partition of each node is about 33MB in size, and the number of partitions is 8, i.e., $n = 8$. The number of items $N = 1000$ and the number of maximal potentially large itemsets $|L| = 1000$. Table 1 shows the databases used and their properties. In it, D_i is the number of transactions in each partitions, T is the average size of the transactions, and I is the average size of the itemsets. The minimum support threshold is 1% while 2% at the last two cases. We ran all CD-SM, PSM and HSM on the 5 databases. Experiments were repeated multiple times to obtain stable values.

Table 1. DATABASE PROPERTIES

Name	D_i	T	I
D1000K.T5.I2	1000K	5	2
D700K.T10.I2	700K	10	2
D700K.T10.I4	700K	10	4
D400K.T20.I4	400K	20	4
D400K.T20.I6	400K	20	6

B. Relative Performance

Fig. 3 shows the response times for the three parallel algorithms on the five databases. Both HSM and PSM are faster than CD-SM in all cases. It seems that the response time of the HSM is near a constant value. But this is a mere coincidence due to the adjustment on the experiment parameters including transaction number, transaction average size and the minimum support threshold.

Fig. 4 shows the pruning effects. It is the ratio of the number of candidate sets with pruning over that generated by *Apriori_gen* only. There are much less candidate

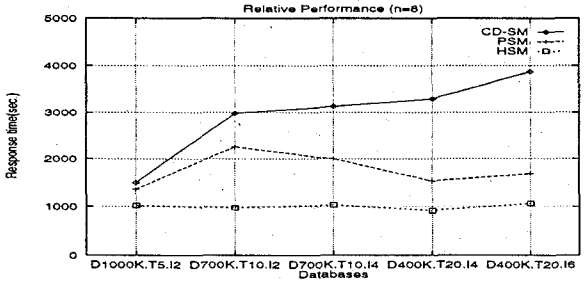

Fig. 3. Relative Performance

itemsets generated when *distributed pruning* and *global pruning* techniques are used. It is obviously that the pruning effect is related to the data distribution among the database partitions. The expected results is that the more data skewness among the partitions, the better the pruning effect. For an extreme example, let 2 database partitions for 2 processors, if itemsets AB, AC, BC are all *gl_large* at both 2 processors, then neither *distributed pruning* nor *global pruning* can prune the itemset ABC out. Extensive studies on the skewness as a parameter on the pruning effect has been performed and will be reported in the future.

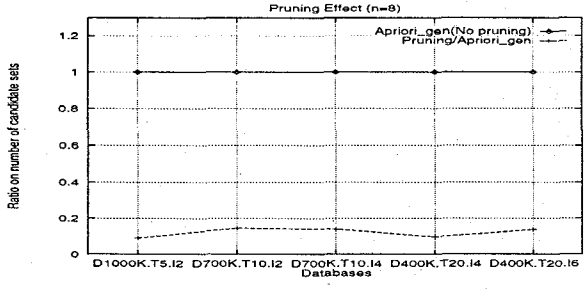

Fig. 4. Pruning Effect

C. Parallel Performance

We have investigated the performance speedup on a fixed size database with increasing number of processors and partitions. The database D700K.T10.I4 was chosen as the dataset with the minimum support threshold 1.0%. Fig. 5 presents the relative speedup. The result is very encouraging. Both HSM and PSM performed better speedup than CD-SM. Especially, HSM has achieved a superlinear speedup. The reason is that the pruning effect is augmented when the number of partitions is increased. Although there was the same pruning effect in PSM algorithm, it didn't present a superlinear property because there is no optimization on I/O reduction.

Another phenomenon in Fig. 5 is that speedup of the three algorithms are not linear. The reason is that the

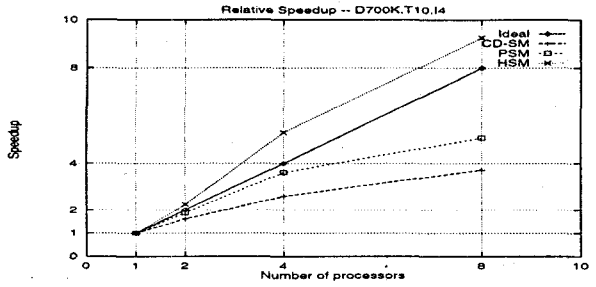

Fig. 5. Speepup

I/O mechanism of SGI Power challenge is not parallel. The I/O contention among processors increase when the number of processor increases, and hence has a negative impact on the performance.

VI. CONCLUSIONS

In this paper, we proposed two parallel algorithms PSM and HSM for mining association rules on the SGI Power Challenge shared-memory multi-processor. PSM is incorporated with two candidate pruning techniques, *distributed pruning* and *global pruning*. HSM further enhances PSM by utilizing an I/O reduction strategy. The experiments showed that both algorithms performed better than CD-SM. In the future work, we are interested in using dynamic candidates generation approach to further reduce the I/O cost and adopt an asynchronous mechanism on shared-momory parallel system to speedup the response time.

References

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

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, Santiago, Chile, 1994, pp. 487–499.

[3] R. Agrawal and J.C. Shafer. Parallel mining of association rules: Design, implementation and experience. Special Issue in Data Mining, *IEEE Trans. on Knowledge and Data Engineering*, IEEE Computer Society, V8, N6, December 1996, pp. 962–969.

[4] S. Brin, R. Motwani, J. Ullman, S. Tsur. Dynamic itemsets counting and implication rules for market basket data. In *Proc. of 1997 ACM-SIGMOD Int. Conf. On Management of Data*, 1997.

[5] D. W. Cheung, J. Han, V. T. Ng, A. W. Fu, Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. of 4th Int. Conf. on Parallel and Distributed Information Systems*, Miami Beach, Florida, December, 1996, pp. 31–43.

[6] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. Fu. Efficient Mining of Association Rules in Distributed Databases. Special Issue in Data Mining, *IEEE Trans. on Knowledge and Data Engineering*, IEEE Computer Society, V8, N6, December 1996, pp. 911–922.

[7] E. Han, G. Karypis and V. Kumar. Scalable parallel data mining for association rules. In *Proc. of 1997 ACM-SIGMOD Int. Conf. On Management of Data*, 1997.

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

[9] J. S. Park, M. S. Chen, and P. S. Yu, Efficient parallel mining for association rules. In *Proc. of the 4th Int. Conf. on Information and Knowledge Management*, Baltimore, Maryland, 1995, pp. 31-36.

[10] T. Shintani, M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc. of 4th Int. Conf. on Parallel and Distributed Information Systems*, 1996.

[11] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li, Parallel data mining for association rules on shared-memory multi-processors. *Supercomputing'96*, Pittsburg, PA, Nov 17-22, 1996.