Efficient Mining of Association Rules in Distributed Databases *

David W. Cheung† Vincent T. Ng‡ Ada W. Fu†† Yongjian Fu‡‡

† Department of Computer Science, The University of Hong Kong, Hong Kong.

‡ Department of Computing, Hong Kong Polytechnic University, Hong Kong.

†† Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

‡‡ School of Computing Science, Simon Fraser University, Burnaby, Canada.

Abstract

Many sequential algorithms have been proposed for mining of association rules. However, very little work has been done in mining association rules in distributed databases. A direct application of sequential algorithms to distributed databases is not effective, because it requires a large amount of communication overhead. In this study, an efficient algorithm, DMA, is proposed. It generates a small number of candidate sets and requires only $O(n)$ messages for support count exchange for each candidate set, where n is the number of sites in a distributed database. The algorithm has been implemented on an experimental test bed and its performance is studied. The results show that DMA has superior performance when comparing with the direct application of a popular sequential algorithm in distributed databases.

Index terms : data mining, knowledge discovery, distributed data mining, association rule, distributed database, distributed algorithm, partitioned database.

1 Introduction

Database mining has recently attracted tremendous amount of attention in database research because of its applicability in many areas, including decision support, marketing strategy and financial forecast. The research community has observed that data mining, together with data warehousing and data repositories are three new uses of database technology, which are considered as important areas in database research [20].

Many interesting and efficient data mining algorithms have been proposed (e.g., see [2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 15, 16, 17, 19, 21]). These database-oriented mining algorithms can be classified

*The research of the authors were supported in part by RGC (the Hong Kong Research Grants Council) grant 338/065/0026.

into two categories: *concept generalization-based discovery* and *discovery at the primitive concept levels*. The former relies on the generalization of concepts (attribute values) stored in databases. One such example is the DBMiner system [7, 12]. The latter discovers strong regularities (rules) from the database without concept generalization. Association rule [4, 6, 16] is an important type of rules in the latter approach.

Most of the algorithms for mining association rules proposed so far are sequential algorithms. An algorithm PDM has been proposed recently for parallel mining of association rules [17]. It is an adaptation of the DHP algorithm in the parallel environment [16]. Another algorithm Count Distribution (CD), which is an adaptation of the Apriori algorithm, has also been proposed for the same parallel mining environment with an implementation on the IBM SP2 [5]. To the best of our knowledge, very little work has been done on the mining of association rules in a distributed database environment. In this paper, we have developed a distributed algorithm DMA (D*istributed* M*ining of* A*ssociation rules*), which can be used to solve this problem.

The distributed database in our model is a horizontally partitioned database. The database schema of all the partitions are the same, i.e., their records are transactions on the same set of items. (DMA can be modified for the case in which the schema at different sites are not completely identical.) Many distributed databases are horizontally partitioned. For example, a retail chain may have several regional data centers, each manages the transaction records in its own region. It is important to mine the association rules based on data from all the centers. Distributed mining can be applied to many applications which have their data sources located at different places.

In the sequential environment, many algorithms have been proposed for mining association rules. The most popular are the Apriori, DHP, and PARTITION algorithms [6, 16, 19]. A candidate set generation function Apriori-gen is adopted in the Apriori algorithm which supports an efficient method for candidate set generation. DHP applies a hashing technique to prune away some size-2 candidate sets to improve its efficiency. PARTITION divides the database into small partitions such that they can be processed efficiently in memory independently to find out their large itemsets. The large itemsets from the partitions are then combined to form a set of candidate sets. Following that, only one scan of the database is required to find out the large itemsets from the candidates.

In the parallel environment, the PDM algorithm proposed in [17] tries to parallelize the DHP algorithm. Each node computes the *globally large itemsets* by exchanging their *support counts* (or *counts*, as referred in some literatures) of the *candidate sets*. In order to apply the hashing technique, all nodes have to broadcast the hashing result, which causes a huge amount of communication. In [17], a technique has been proposed to decrease the number of messages. Amount all the hash buckets, only those in which the total count are larger than a threshold are selected for bucket count exchange, so that not all buckets have to be broadcasted. After a node receives these partial count for the selected buckets, it polls the other sites to get the total counts. However, there are two unfavourable features in this proposal. Firstly, the reduction of candidate sets is only done in the second iteration. The number of candidate sets in some other iterations could also be quite large. Secondly, to find the large candidate sets, $O(n^2)$ messages are required for support count exchange for each candidate set, where n is the number of nodes.

Another algorithm proposed for parallel mining of association rules is the CD algorithm [5]. It is an adaptation of the Apriori algorithm in the parallel case. At each iteration, it generates the candidate sets at every site by applying the Apriori-gen function on the set of large itemsets found at the previous iteration. Every site then computes the local support counts of all these candidate sets and broadcasts them to all the other sites. Subsequently, all the sites can find the globally large itemsets for that iteration, and then proceed to the next iteration. This algorithm has a simple communication scheme for count exchange. However, it also has the similar problems of higher number of candidate sets and larger amount of communication overhead.

The efficiency of the algorithm DMA that we have developed is attributed mainly to the following two features.

1. Both Apriori and DHP generate the candidate sets by applying the Apriori-gen function on the large itemsets found in the previous iteration. CD and PDM use the same technique in the parallel environment. DMA uses a new technique to generate a much smaller set of candidate sets then either Apriori or DHP. (This will be explained in Section 3.2).

2. In DMA, to determine whether a candidate set is large, only $O(n)$ messages are needed for support count exchange. This is much less than a straight adaptation of Apriori, which requires $O(n^2)$ messages for support count exchange.

Distributed database has an intrinsic *data skewness property*. The distribution of the itemsets in different partitions are not identical, and many items occur more frequently in some partitions than the others. For example, in a distributed database of a national supermarket chain, it is expected the consumers' purchasing patterns in New York City will be quite different from that in Los Angles. As a result, many itemsets may be large locally at some sites but not necessarily in the other sites. This skewness property poses a new requirement in the design of mining algorithm.

Furthermore, DMA can be applied to the mining of association rules in a large centralized database by partitioning the database to the nodes of a distributed system. This is particularly useful if the data set is too large for sequential mining.

Extensive experiments have been conducted to study the performance of DMA and compare it against the algorithm Count Distribution (CD), which is a direct application of the Apriori algorithm to distributed databases. The remaining of the paper is organized as follows. A brief summary of mining association rules in the sequential environment will be discussed in Section 2. In Section 3, the problem of mining association rules in a distributed database is defined and some important results are discussed. The algorithm DMA is presented in Section 4. A performance study is discussed in Section 5. Some discussion and conclusions are presented in Sections 6 and 7.

2 Sequential Mining of Association Rules

2.1 Association rules

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of *items*. Let DB be a database of transactions, where each transaction T is a set of items such that $T \subseteq I$. Given an *itemset* $X \subseteq I$, a transaction T *contains* X if and only if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$. The association rule $X \Rightarrow Y$ holds in DB with *confidence* c if $c\%$ of the transactions in DB that contain X also contain Y. The association rule $X \Rightarrow Y$ has *support* s in DB if $s\%$ of the transactions in DB contain $X \cup Y$.

Given a minimum confidence threshold *minconf* and a minimum support threshold *minsup*, the problem of mining association rules is to find all the association rules whose confidence and support are larger than the respective thresholds. We also call an association rule a *strong* rule to distinguish it from the *weak* ones, i.e., those that do not meet the thresholds [13].

For an itemset X, its *support* is defined similarly as the percentage of transactions in DB which contains X. We also use $X.sup$, to denote its *support count*, which is the number of transactions in DB containing X. Given a minimum support threshold *minsup*, an itemset X is *large* if its support is no less than *minsup*. Moreover, for presentation purpose, we will call an itemset of size-k a k-itemset. It has been shown that the problem of mining association rules can be reduced to two subproblems [4].

1. Find all large itemsets for a pre-determined minimum support.

2. Generate the association rules from the large itemsets found.

The most crucial factor that affects the performance of mining association rules is to find efficient method to resolve the first problem [6].

2.2 Apriori algorithm

The Apriori algorithm is one of the most popular algorithm in the mining of association rules in a centralized database. The main idea of Apriori is outlined in the following [6].

1. The large itemsets are computed through iterations. In each iteration, the database is scanned once and all large itemsets of the same size are computed. The large itemsets are computed in the ascending order of their sizes.

2. In the first iteration, the size-1 large itemsets are computed by scanning the database once. Subsequently, in the k-th iteration ($k > 1$), a set of candidate sets C_k is created by applying the candidate set generating function Apriori-gen on L_{k-1}, where L_{k-1} is the set of all large $(k-1)$-itemsets found in iteration $k-1$. Apriori-gen generates only those k-itemset whose every $(k-1)$-itemset subset is in L_{k-1}. The support counts of the candidate itemsets in C_k are then computed by scanning the database once and the size-k large itemsets are extracted from the candidates.

Two interesting extensions of the Apriori algorithm are the DHP [17] and PARTITION algorithms [19]. In the first iteration, while it is computing the support counts of the size-1 itemsets, DHP stores the support counts of the size-2 candidate itemsets in a hash table. Upper bounds of the support counts of the size-2 candidates can be deduced from the hash table and are used to prune away some size-2 candidates in the second iteration. As a result of the hashing and pruning, the cost of computing the support counts of the size-2 candidate sets is reduced substantially in DHP.

The PARTITION algorithm divides the database into partitions such that each of them can be processed efficiently in memory to find the itemsets which are large in it. The set consists of all these itemsets becomes a candidate set for finding the large itemsets in the database. The advantage of the PARTITION algorithm is that only one scan of the database is required after the candidate sets are found in the partitions.

3 Mining of Association Rules in Distributed Databases

3.1 Problem Description

Let DB be a partitioned database located at n sites $S^1, S^2, \ldots, S^n$. The database partitions at these sites are $\{DB^1, DB^2, \ldots, DB^n\}$. (In the following, we will adopt the convention of attaching a superscript i on a notation to denote the corresponding distributed notation for site S^i.)

Let the size of DB and the partitions DB^i be D and D^i, respectively. For a given itemset X, let $X.sup$ and $X.sup^i$ be the respective support counts of X in DB and DB^i. We will call $X.sup$ the *global support count* and $X.sup^i$ the *local support count* of X at site S^i. For a given minimum support s, X is *globally large* if $X.sup \geq s \times D$; correspondingly, X is *locally large* at site S^i, if $X.sup^i \geq s \times D^i$. In the following, we will use L to denote all the globally large itemsets in DB and L_k to denote all globally large k-itemsets in L. The problem of mining association rules in a distributed database DB can be reduced to the finding of all globally large itemsets.

3.2 Generate a Smaller Set of Candidate Sets

Before we discuss how to generate a small set of candidate sets, we first present a few interesting and useful observations. First of all, we have found that many candidate sets generated by applying the Apriori-gen function are not needed in the search of large itemsets. In fact, there is a natural and effective method for every site to generate its own set of candidate sets, which is typically much smaller than the set of all the candidate sets. Following that, every site only needs to find the large itemsets among these candidate sets. By using this technique, we have achieved an effective division of the mining task amongst the sites in the database. In the following, several lemmas and theorem are described to illustrate the above observations.

Lemma 1 *If an itemset X is locally large at a site S^i, then all its subsets are also locally large at site S^i.*

Proof. This follows from the definition of locally large. □

A similar result as Lemma 1 for centralized database first appeared in [4].

Lemma 2 *If an itemset X is globally large, then there exists a site S^i, $(1 \leq i \leq n)$, such that X and all its subsets are locally large at site S^i.*

Proof. If $X.sup^i < s \times D^i$ for all $i = 1, \ldots, n$, then $X.sup < s \times D$, and X cannot be globally large. Therefore, X must be locally large at some site S^i. It follows from Lemma 1 that all the subsets of X must be locally large at S^i. □

For a site S^i, if an itemset X is both locally large at site S^i and globally large, then we say that X is *heavy* at site S^i. We use HL^i to denote the set of heavy itemsets at site S^i, and HL^i_k to denote the set of heavy k-itemsets at site S^i. In DMA, the heavy itemsets at each site play an important role in the generation of candidate sets.

Lemma 3 *If an itemset X is globally large, then there exists a site S^i, $(1 \leq i \leq n)$, such that X is heavy at site S^i.*

Proof. Since X is globally large, it follows from Lemma 2 that X must be locally large at some site S^i, $(1 \leq i \leq n)$. Hence, X is heavy at site S^i. □

Lemma 4 *If an itemset X is heavy at a site S^i, $(1 \leq i \leq n)$, then all its subsets are also heavy at site S^i.*

Proof. If X is heavy at site S^i, then it must be globally large, therefore, all its subsets are globally large. Moreover, since X is locally large at site S^i, it follows from Lemma 1 that all the subsets of X must be locally large at site S^i. Hence, all its subsets are heavy at site S^i. □

Lemma 4 is a very interesting property, it shows that the heavy itemsets at each site have a monotonic subset relationship among them. This relationship also exists among the large itemsets in the centralized case, and it is a necessary condition such that large itemsets can be computed iteratively.

Lemma 5 *If $X \in L_k$, (i.e. X is a globally large k-itemset), then there exists a site i, $(1 \leq i \leq n)$, such that X and all its size $(k-1)$ subsets are heavy at site S^i.*

Proof. This follows from Lemmas 3 and 4. □

Lemma 5 is equivalent to the combination of Lemma 3 and Lemma 4. It is a basis to design an effective method to generate a smaller set of candidate sets in the distributed environment.

In general, in a straightforward adaptation of Apriori, in the k-th iteration, the set of candidate sets would be generated by applying the Apriori-gen function on L_{k-1}. We denote this set of candidate sets by CA_k, (which stands for size-k candidate sets from Apriori). In order words,

$$CA_k = \text{Apriori\_gen}(L_{k-1}).$$

At each site S^i, let CH^i_k be the set of candidates sets generated by applying Apriori-gen on HL^i_{k-1}, i.e.,

$$CH^i_k = \text{Apriori\_gen}(HL^i_{k-1}),$$

(CH stands for candidate sets generated from heavy itemsets). Hence CH^i_k is generated from HL^i_{k-1}, which is only a subset of L_{k-1}.

According to Lemma 5, for every large itemset $X \in L_k$, there exists a site S^i, such that all the size-(k-1) subsets of X are heavy at site S^i; hence $X \in CH^i_k$ for some site S^i. Therefore

$$L_k \subseteq \bigcup_{i=1}^{n} CH^i_k = \bigcup_{i=1}^{n} \text{Apriori\_gen}(HL^i_{k-1}).$$

We use CH_k to denote the set $\cup_{i=1}^{n} CH^i_k$.

Theorem 1 *For every $k > 1$, the set of all large k-itemsets L_k is a subset of $CH_k = \bigcup_{i=1}^{n} CH_k^i$, where $CH_k^i = \text{Apriori\_gen}(HL_{k-1}^i)$. Hence CH_k is a set of candidate sets for the size-k large itemsets.*

Proof. The proof follows from Lemma 5 and the above discussion. □

Since every HL_{k-1}^i in Theorem 1 is a subset of L_{k-1}, the number of candidate sets in CH_k is in general smaller than that in CA_k. In DMA, we use the result in Theorem 1 to generate a set of candidate sets CH_k^i for each site S^i in each iteration. It can be seen that this set of candidate sets is typically much smaller than that in a direct application of Apriori-gen on L_k.

In the following, Example 1 is used to illustrate the reduction of candidate sets by using Theorem 1.

Example 1 Assuming there are 3 sites in a database DB with partitions DB^1, DB^2 and DB^3. After the first iteration, suppose the set of large 1-itemsets $L_1 = \{A, B, C, D, E, F, G\}$, in which A, B, C are locally large at site S^1, B, C, D are locally large at site S^2, and E, F, G are locally large at site S^3. Therefore, $HL_1^1 = \{A, B, C\}$, $HL_1^2 = \{B, C, D\}$, and $HL_1^3 = \{E, F, G\}$.

It follows from Theorem 1 that the set of size-2 candidate sets at site S^1 is equal to CH_2^1, where $CH_2^1 = \text{Apriori\_gen}\ (HL_1^1) = \{AB, BC, AC\}$. Similarly, $CH_2^2 = \{BC, CD, BD\}$, and $CH_2^3 = \{EF, FG, EG\}$. Hence , the set of candidate sets for large 2-itemsets is $CH_2 = CH_2^1 \cup CH_2^2 \cup CH_2^3$, and it only has 8 candidates.

However, if Apriori-gen is applied to L_1, the set of candidate sets $CA_2 = \text{Apriori\_gen}(L_1)$ would have 21 candidates. This shows that the technique in Theorem 1 is very effective in reducing the candidate sets.

□

3.3 Local Pruning of Candidate Sets

In the previous subsection, we have shown that the set CH_k is typically a much smaller set of candidate sets than CA_k. To find the globally large itemsets, subsequent to the generation of CH_k, support count exchange should be done. However, we have observed that some candidate sets in CH_k can be pruned away by using some local information before the count exchange starts.

From Lemma 5, if X is a globally large k-itemset, then there must exist a site S^i, such that $X \in CH_k^i$ and X is heavy at site S^i. As a consequence, X must be locally large at site S^i. Therefore, a site S^i can prune away those candidates in CH_k^i which are not locally large at S^i. In other words, to compute all the large k-itemsets, at each site S^i, DMA can restrict its search domain on all the sets $X \in CH_k^i$ which are locally large at site S^i. For convenience, we use LL_k^i to denote those candidate sets in CH_k^i which are locally large at site S^i.

Follows from the above discussion, in every iteration, (loop counter = k), DMA computes the heavy k-itemsets at each site S^i according to the following procedure.

1. Candidate Sets Generation: generate the candidate sets $CH_k^i = \text{Apriori\_gen}(HL_{k-1}^i)$, based on the heavy itemsets found at site S^i in the $k-1$ iteration. (By doing so, each site actually is responsible for generating its own set of candidate sets, and hence computing its own set of large itemsets.)

2. Local Partition Scanning: for each $X \in CH_k^i$, scan the partition DB^i to compute the local support count $X.sup^i$.

3. Local Pruning: for each $X \in CH_k^i$, if X is not locally large at site S^i, then it is pruned away; the remaining candidate sets are stored in LL_k^i. (The above pruning only removes X from the candidate set at site S^i. X could still be a candidate set at some other site.)

4. Support Count Exchange: broadcast the candidate sets in LL_k^i to other sites to collect support counts; compute their global support counts and find all the heavy k-itemsets in site S^i. (A site S^j, $(j \neq i)$, which has received a request from S^i for support counts, does not need to scan its partition again to compute the support counts. The counts can be computed in advance in Step 2. A detail discussion of this is in Section 4.1).

5. Broadcast Mining Result: broadcast the heavy k-itemsets found to all the other sites.

In the following, we extend Example 1 to Example 2 to illustrate the execution of the above procedure. Before that, for clarity purpose, we list the notations used so far in our discussion in Table 1.

D	The number of transactions in database DB
s	The support threshold $minsup$
L_k	The set of globally large k-itemsets
CA_k	The set of candidate sets generated from L_k
$X.sup$	The global support count of an itemset X
D^i	The number of transactions in the partition DB^i
HL_k^i	The set of heavy k-itemsets at site S^i
CH_k^i	The set of candidate sets generated from HL_{k-1}^i
LL_k^i	The set of locally large k-itemsets in CH_k^i
$X.sup^i$	The local support count of an itemset X at site S^i

Table 1: Notation Table.

Example 2 In Example 1, assume the database has 150 transactions and each one of the 3 partitions has 50 transactions. Also assume that the support threshold $s = 10\%$. Moreover, as has been illustrated in Example 1, in the second iteration, the candidate sets generated at site S^1 are $CH_2^1 = \{AB, BC, AC\}$; at site S^2 are $CH_2^2 = \{BC, BD, CD\}$; and at site S^3 are $CH_2^3 = \{EF, EG, FG\}$.

In order to compute the large 2-itemsets, DMA first computes the local support counts at each site. The result is recorded in Table 2. The last three rows are the local support counts of the candidate sets at the corresponding sites. For example, the candidate sets at site S^1 are listed in the first column, and their local support counts are listed in the second column.

From Table 2, it can be seen that $AC.sup^1 = 2 < s \times D^1 = 5$, therefore, AC is not locally large. Hence, the candidate set AC is pruned away at site S^1. On the other hand, both AB and BC

S^1		S^2		S^3	
CH_2^1	$X.sup^1$	CH_2^2	$X.sup^2$	CH_2^3	$X.sup^3$
AB	5	BC	10	EF	8
BC	10	CD	8	FG	3
AC	2	BD	4	EG	3

Table 2: Locally Large Itemsets.

have enough local support counts and they survive the local pruning. Hence $LL_2^1 = \{AB, BC\}$. Similarly, BD is pruned away at site S^2 and $LL_2^2 = \{BC, CD\}$. The only remaining candidate set at site S^3 is EF, i.e., $LL_2^3 = \{EF\}$. After the local pruning, the number of size-2 candidate sets has been reduced to half of the original size.

Once the local pruning is completed, each site broadcasts messages containing all the remaining candidate sets to the other sites to collect their support counts. The result of this count support exchange is recorded in Table 3.

locally large candidate sets	request broadcast from sites	$X.sup^1$	$X.sup^2$	$X.sup^3$	$X.sup$
AB	S^1	5	4	4	13
BC	S^1, S^2	10	10	2	22
CD	S^2	4	8	4	16
EF	S^3	4	3	8	15

Table 3: Globally Large Itemsets.

The request for support count for AB is broadcast from S^1 to site S^2 and S^3, and the counts sent back are recorded at site S^1 as in the second row of Table 3. The other rows record similar count exchange activities at the other sites. At the end of the iteration, site S^1 finds out that only BC is heavy, because $BC.sup = 22 > s \times D = 15$, and $AB.sup = 13 < s \times D = 15$. Hence the heavy 2-itemset at site S^1 is $HL_2^1 = \{BC\}$. Similarly, $HL_2^2 = \{BC, CD\}$ and $HL_2^3 = \{EF\}$. After the broadcast of the heavy itemsets, all sites return the large 2-itemsets $L_2 = \{BC, CD, EF\}$.

In terms of message communication, in this example, most of the candidate sets are locally large at one site. For each one of them, only one broadcast and receive are needed. However, for the candidate set BC, messages are broadcast from both S^1 and S^2, which is not as efficient as in the single broadcast case. In Section 3.4, an optimization technique to eliminate this duplication will be discussed.

$\square$

3.4 Message Optimization for Finding Large Itemsets

In a straight adaptation of the sequential Apriori algorithm, not only the number of candidate sets generated is larger, but the number of messages for count exchange for each candidate set is also large. This is due to the broadcast for every candidate set from all the sites. This requires $O(n^2)$ messages in total for each candidate set, where n is the number of partitions.

In DMA, if a candidate set X is locally large at a site S^i, S^i only needs $O(n)$ messages to collect all the support counts for X. In general, very few candidate sets are locally large at all the sites. Because of the data skewness property, the percentage of overlappings of the locally large candidate sets from different sites should be small. Therefore, in most cases, DMA requires much less than $O(n^2)$ messages for each candidate set.

To ensure that DMA requires only $O(n)$ messages for every candidate set in all cases, an optimization technique has been introduced. To achieve single broadcast, DMA uses some simple assignment functions, which could be a hash function, to determine a *polling site* for each candidate set.

For each candidate set X, its polling site is responsible for broadcasting the polling request, collecting the support counts, and determine whether X is large. Since there is only one polling site for each candidate set X, the number of messages required for count exchange for X is $O(n)$.

In the k-th iteration, after the local pruning phase has been completed at a site S^i, DMA uses the following procedure to do the polling.

1. Candidates sent to Polling Sites: S^i acts as a home site of its candidate sets; for every polling site S^j, S^i finds all the candidate sets in LL_k^i whose polling site are S^j and stores them in $LL_k^{i,j}$, (i.e., candidates are being divided into groups according to their polling sites), the local support counts of the candidate sets are also stored in the corresponding set $LL_k^{i,j}$; sends each $LL_k^{i,j}$ to the corresponding polling site S^j.

2. Polling Site send Polling Requests: S^i acts as a polling site; S^i receives all $LL_k^{j,i}$ sent to it from the other sites; for every candidate set X received, S^i finds the list of originating sites from which X is being sent; S^i then broadcasts the polling requests to the other sites not on the list to collect the support counts.

3. Remote Site reply Polling Requests: S^i acts as a remote site to reply polling requests sent to it; for every polling request $LL_k^{p,i}$ from polling site S^p, S^i sends the local support counts of the candidates in $LL_k^{p,i}$ back to S^p. (There is no need to scan the partition D^i again to find the local support counts. It is found already during the local pruning. Please see Section 4.1 for details.)

4. Polling Site Compute Heavy Itemsets: S^i acts as a polling site to compute the heavy itemsets; S^i receives the support counts from the other sites; computes the global support counts for its candidates in LL_k^i and finds the heavy itemsets; eventually, S^i broadcasts the heavy itemsets together with their global support counts to all the sites.

Example 3 In Example 2, assuming that S^1 is assigned as the polling site of AB and BC, S^2 is assigned as the polling site of CD, and S^3 is assigned as the polling site of EF.

Following from the assignment, site S^1 is responsible for the polling of AB and BC. In the simple case of AB, S^1 sends polling requests to S^2 and S^3 to collect the support counts. As for BC, it is locally large at both S^1 and S^2, the pair $\langle BC, BC.sup^2\rangle = \langle BC, 10\rangle$ is sent to S^1 by S^2. After S^1 receives the message, it sends a polling request to the remaining site S^3. Once the support count $BC.sup^3 = 2$ is received from S^3, S^1 finds out that $BC.sup = 10 + 10 + 2 = 22 > 15$. Hence BC is a heavy itemset at S^1. By using a polling site, DMA has eliminated the double polling messages for BC.

□

4 Algorithm for Distributed Mining of Association Rules

In this section, we present the DMA algorithm (DMA) in detail based on the above discussion. Before the description of the algorithm, we will discuss a technique for computing the local support counts of all the candidate itemsets at different sites by performing only one single scan on each partition.

4.1 Optimizing Partition Scanning for Count Exchanges

At each site S^i, DMA has to find two sets of support counts in order to do local pruning and count exchange. The first set is the local support counts of all the candidate sets generated at site S^i. (These candidate sets are the sets in CH_k^i described in Theorem 1). A hash tree can be used to store the support counts of these candidate sets [6]. A scan on the partition DB^i is needed to compute the counts to store in the hash tree. On the other hand, in order to answer the polling requests from the other sites, a second set of support counts of the candidate sets generated at the other sites is needed. If these counts are computed after the requests are received, a second scan on the partition is unavoidable.

In order to avoid doing two scans, DMA is required to find the two sets of support counts by one scan on the partition and store the counts on the same hash tree. This is possible because the heavy sets for candidate set generation are available to all the sites at the end of each iteration. According to Theorem 1, at a site S^i, the set of candidate sets generated in the k-th iteration is $CH_k^i = \text{Apriori\_gen}(HL_{k-1}^i)$. On the other hand, those generated in any other site S^j is $CH_k^j = \text{Apriori\_gen}(HL_{k-1}^j)$. Since HL_{k-1}^i and HL_{k-1}^j, $(j = 1, \ldots, n, j \neq i)$, are available at S^i, S^i can compute all these candidate sets and put them in the same hash tree before the scan for their local support counts starts. In other words, every site only needs to scan its partition once to find the local support counts of the itemsets in $CH_k = \cup_{i=1}^n \text{Apriori\_gen}(HL_{k-1}^i)$. With this technique, the two sets of support counts required for local pruning and count exchange can be found in a single scan of the partition. Therefore, the number of scans in DMA is minimized and is comparable to that in the sequential case.

Further more, since every site will have the same set of candidate sets CH_k, there is no need to send the itemset names in a polling request, only their positions in the ordered list of the itemsets in CH_k is required. This would optimize the message size needed for count exchange.

4.2 The DMA algorithm

In this section, we present the DMA algorithm in details.

Algorithm 1 DMA: Distributed Mining of Association rules algorithm

Input: (1) DB^i : the database partition at each site, (its size is equal to D^i); (2) s: the minimum support threshold; both submitted at each site S^i, $(i = 1, \ldots, n)$;

Output: L: the set of all large itemsets in DB, returned at every site;

Method: iterates the following program fragment distributively at each site S^i starting from $k = 1$, where k is the iteration loop counter; the algorithm terminates when either L_k returned is empty or the set of candidate sets CH_k is empty

```
/* Local Pruning */
if k = 1 then
  scan DB^i to compute T^i_1;
  /* T^i_1 is an array containing all size-1 itemsets in DB^i and */
  /* their local support counts in site S^i */
else {
  CH_k = ∪_{i=1}^n CH^i_k = ∪_{i=1}^n Apriori_gen(HL^i_{k-1});
  /* generate size-k candidate sets */
  scan DB^i to built the hash tree T^i_k; }
  /* T^i_k contains all candidate sets in CH_k and */
  /* their support counts in site S^i */
for_all X ∈ T^i_k do
  if X.sup^i ≥ s × D^i then
    for j = 1 to n do
      if polling_site(X) = S^j then add ⟨X, X.sup^i⟩ into LL^{i,j}_k;
/* compute the locally large candidates and divide them according to their polling sites */

/* Send Candidates to Polling Sites */
for j = 1, ..., n do
  send LL^{i,j}_k to site S^j;

/* Receive Candidates as a Polling Site */
for j = 1, ..., n do {
  receive LL^{j,i}_k;
  for_all X ∈ LL^{j,i}_k do {
    store X in LP^i_k;
    update X.large_sites in LP^i_k to record the sites at which X is locally large; }
}

/* Send Polling Requests as a Polling Site to Collect Support Counts */
for_all X ∈ LP^i_k do {
```

broadcast polling requests for X to the sites S^j, where $S^j \notin X.large\_sites$;
receive $X.sup^j$ from the sites S^j, where $S^j \notin X.large\_sites$;
};

/* Compute Global Support Counts and Heavy Itemsets
for_all $X \in LP_k^i$ do {
$X.sup = \sum_{i=1}^{n} X.sup^i$;
if $X.sup \geq s \times D$ then insert X into H_k^i;
/* filter out the heavy k-itemsets;
};
broadcast H_k^i;
receive H_k^j from all other sites S^j, $(j \neq i)$;
return $L_k = \cup_{i=1}^{n} H_k^i$.

□

5 Performance Study of DMA

We have done an in-depth performance study on DMA to confirm our analysis of its efficiency. DMA is implemented on a share-nothing distributed system by using PVM (Parallel Virtual Machine) [11]. A 10Mb LAN is used to connect six RS/6000 workstations running the AIX system to perform the study. The database in the experiment is composed of synthetic data.

In order to study the performance of DMA, we have also implemented the algorithm CD in our test bed. In each iteration, CD generates the candidate sets at every site by applying the Apriori-gen function on the set of large itemsets found in the previous iteration. Every site computes the local support counts of all these candidate sets and broadcasts them to the other sites. All the sites can then find the globally large itemsets for that iteration.

We have performed two experiments to compare the performance of DMA and CD. In the first experiment, the test bed has a fixed number of sites. The aim is to perform the comparison with respect to different support thresholds and database sizes. In the second experiment, the threshold and database size are fixed, and the performance of the two algorithms are compared with respect to different number of sites. The result of the first experiment is described in detail in Section 5.1, and those of the second experiment is presented in Section 5.2.

The databases used in our experiments are synthetic data generated using the same techniques introduced in [6, 16]. The parameters used are similar to those in [16]. Table 4 is a list of the parameters and their values used in our synthetic databases. Readers not familiar with these parameters can refer to [6, 16]. In the following, we use the notation Tx.Iy.Dm to denote a database in which $D = m$ (in thousands), | T | = x, and | I | = y.

5.1 Performance Comparison with Different Thresholds and Database Sizes

In the first experiment, the test bed consists of three sites. The purpose of this experiment is to compare the performance between DMA and CD with respect to different thresholds and database

Parameter	Interpretation	Value
D	The number of transactions in database DB	
$\mid$ T $\mid$	Mean size of the transactions	10
$\mid$ I $\mid$	Mean size of the maximal potentially large itemsets	4
$\mid$ L $\mid$	Number of potentially large itemsets	2000
N	Number of items	1000
S_q	Clustering size	5 - 6
P_s	Pool size	50 - 70
c_r	Correlation level	0.5
M_f	Multiplying factor	1260 - 2400

Table 4: Parameter Table.

sizes. Each site has its own local disk, and its partition is loaded on its local disk before the experiments start.

The three partitions are generated separately using the parameters and the values in Table 4. In order to control the skewness of the partitions, two more control parameters are introduced. These two parameters are *primary range* r_p and *secondary range* r_s. The primary range is an interval of items, and the secondary range is a sub-interval of the primary range. If the items range from 1 to 1000, a possible pair of primary and secondary ranges could be $r_p = [1, 1000]$, and $r_s = [1, 700]$. As described in [16], itemsets are generated as groups of similar itemsets. The size of each group is controlled by the clustering size S_q, and the size of the itemsets is a Poisson distribution. In our synthesizing model, the first itemset in a group is picked randomly from the primary range r_p, and the other itemsets in the group contain two parts, the *head* and the *tail*. The head is a random extraction from the first itemset that has been generated. If the head cannot fill up the itemset size, then the tail is picked randomly from the secondary range r_s. By doing this, most itemsets generated are within the primary range, with some clustering in the secondary range. Therefore, we can generate databases that have certain skewness towards the secondary range.

The data skewness of a distributed database can be controlled by using different primary and secondary ranges for different partitions. In Table 5, the primary and secondary ranges of the three partitions in the first experiment are listed. The first two partitions are skewed towards the ranges $[1, 700]$ and $[300, 1000]$ respectively. The third partition DB^3 is generated with two clustering ranges. Two disjoint pools of large itemsets are used in synthesizing DB^3. The first one is from the range pair $[1, 550]$ and $[1, 400]$, while the second one is from the range pair $[450, 1000]$ and $[600, 1000]$. Half of the transactions are picked from the first pool, and the other half from the second pool. Together, these three partitions exhibit a certain degree of skewness.

In this experiment, the sizes of the databases range from 100K to 900K transactions, and the minimum support threshold ranges from 0.75% to 2%. While the number of candidate sets in DMA are different at each site; the number in CD remains the same at all sites.

When comparing DMA against CD, we experienced, on average, a 65% reduction of the number of candidate sets at every site. In Figure 1, the average number of candidate sets generated by

partition	primary range	secondary range
DB^1	$[1, 1000]$	$[1, 700]$
DB^2	$[1, 1000]$	$[300, 1000]$
DB^3	$[1, 550]$	$[1, 400]$
	$[450, 1000]$	$[600, 1000]$

Table 5: Partition Primary and Secondary Ranges .

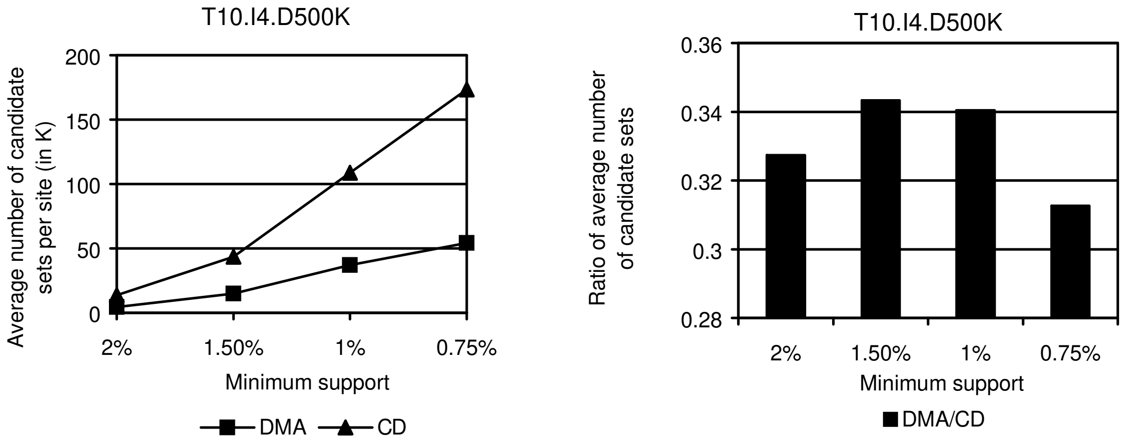

Figure 1: Candidate Sets Reduction.

DMA and CD at each site for a database of size 500K transactions are plotted against the support thresholds. DMA has much less candidate sets in all cases, and the difference increases as the support decreases. For the same database, the ratios of the number of candidate sets between DMA and CD are presented also in Figure 1. The figure shows that the reduction in the number of candidate sets in DMA against CD is about $65\% - 70\%$.

The above comparison is on the number of candidate sets per site. The result has direct implication on the reduction in the total number of messages required, because only one site will generate messages for a candidate set to do polling.

The reduction in the total messages required is bigger than that in candidate sets when comparing DMA against CD. We have experienced a reduction of about 90% in total message size in all cases. In Figure 2, for the database of 500K, the total message size needed by DMA and CD are plotted against the support thresholds. Moreover, the ratios of the total message sizes between DMA and CD are presented in the same figure. The reduction is larger when the support threshold is smaller, (i.e., when there are more large itemsets). In the bar chart of Figure 2, it can be seen that DMA requires $6\% - 12\%$ of the messages of CD.

We have also compared the execution time between DMA and CD. With the database of 500K, DMA is about 7% to 25% faster than CD, depending on the support threshold. In Figure 3, the execution time of DMA and CD are plotted against the thresholds for the 500K database. The ratios of speed-up are presented in the same figure in bar chart. For some other database sizes in this experiment, the best speed-up can reach about 55%.

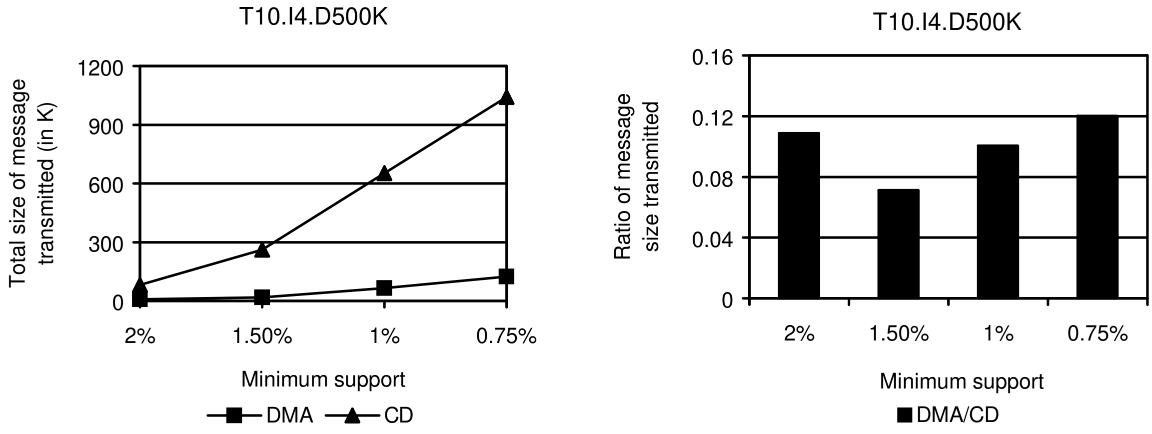

Figure 2: Message Size Reduction.

Even though the speed-up in our experiment is substantial, it does not seem to be as significant as the reduction in message size. The main reason is that the overhead in communication is relatively small in our test bed. If DMA is running in a distributed database, whose partitions are placed in far apart locations, the speed-up will be more significant.

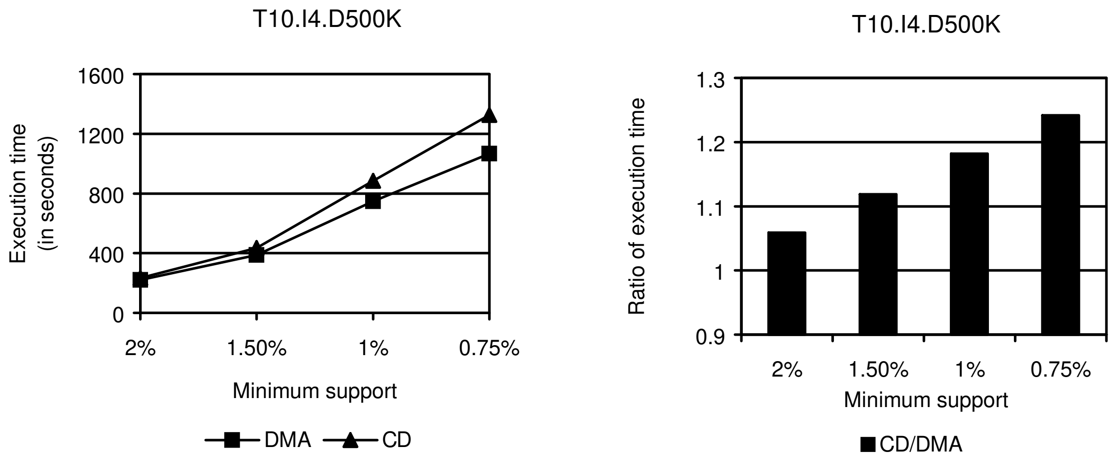

Figure 3: Execution Time Speed Up.

In this experiment, we have also compared DMA against CD on a series of 5 databases from 100K to 900K transactions. In terms of candidate sets and total message size reduction, the improvement in DMA against CD is very steady. In Figure 4, the average number of candidate sets per site in DMA is compared to that in CD over all the 5 databases, for the threshold $s = 0.75\%$. The ratios between them are plotted in the figure. The result shows that the percentage of reduction is about 70% in all cases.

In Figure 5, the total size of message communication in DMA is compared to that in CD over all the 5 databases, for the threshold $s = 0.75\%$. The ratios between them are presented in the figure, and it shows that the reduction is between 88% to 89% in all cases.

In Figure 6, the execution time of DMA is compared to that of CD over all the 5 databases, for the same threshold $s = 0.75\%$. The ratios between them are plotted in the figure and DMA is about 18% to 55% faster than CD.

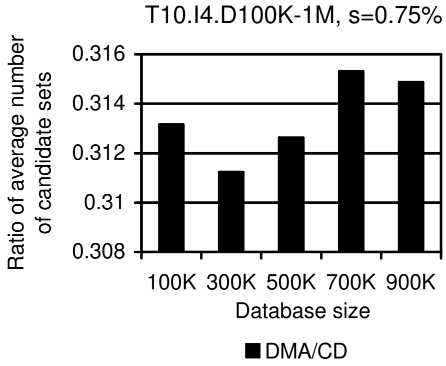

Figure 4: Candidate Sets Reduction.

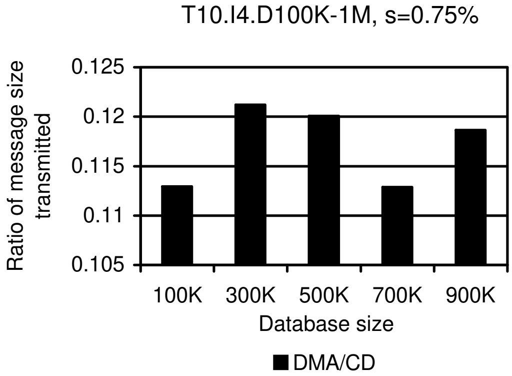

Figure 5: Message Size Reduction.

5.2 Performance Comparison with Different Number of Sites

In the second experiment, the test bed consists of six RS/6000 workstations. The synthetic database is generated similar to that in the first experiment. The aim of this experiment is to compare DMA against CD when the number of sites changes. In the following, we will describe the result of a comparison in which the number of sites varies from three to six. The size of the database is 200K transactions, and it is partitioned equally across all the sites. The minimum support threshold is 3%.

Similar to the first experiment, we found significant reduction in both the number of candidate sets and the total message sizes in all the cases in which the number of sites are 3, 4, 5, and 6, respectively. In Figure 7, the average number of candidate sets per sites is compared between DMA and CD. A reduction of about $75\% - 90\%$ is witnessed in DMA. In Figure 8, the ratios of the total message sizes of the two algorithms is presented. DMA has about $85\% - 90\%$ reduction in message sizes in all the cases. Lastly, the execution time ratios are described in Figure 9, again, DMA is shown to be about $25\% - 35\%$ faster than CD in all the cases.

In general, the performance of DMA depends on the distribution of the data across the partitions. If the itemsets are distributed with a higher skewness among the partitions, the techniques of local pruning and candidate set generation reduction in DMA would be more powerful. When comparing

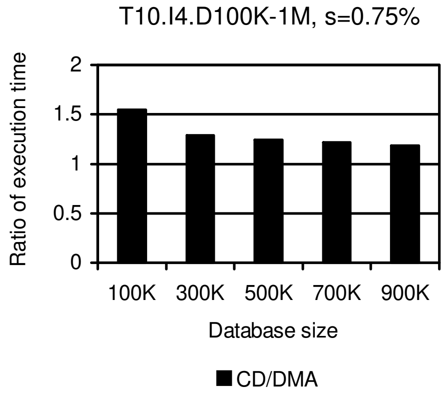

Figure 6: Execution Time Speed Up.

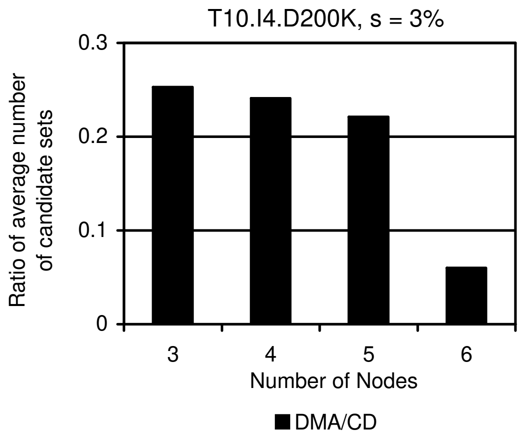

Figure 7: Candidate Sets Reduction (n = 3,4,5,6).

the results of the above two different experiments, it can be observed that DMA performs better when the number of nodes is higher. This could be the consequence of a higher data skewness due to the increased number of partitions.

6 Discussion

The efficiency of DMA is attributed to three techniques: (1) candidate sets generation, (2) local pruning, and (3) messages optimization. In the described DMA, only local information available in each partition is considered in the local pruning. Can we take advantage of the global information available to do more pruning before support count exchange starts ? In fact, at the end of each iteration, the polling site of a candidate set X not only knows the global support count of X but also all the local support counts of X. The set of local support counts can be broadcasted to all the sites together with X at the end of each iteration. We now discuss an optimization technique which makes use of this global information to prune candidate sets.

If X is a k-itemset, with respect to each partition DB^i, $(1 \leq i \leq n)$, we use $maxsup^i(X)$ to

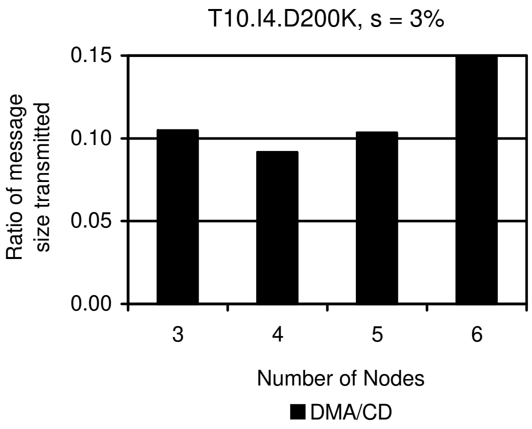

Figure 8: Message Size Reduction (n = 3,4,5,6).

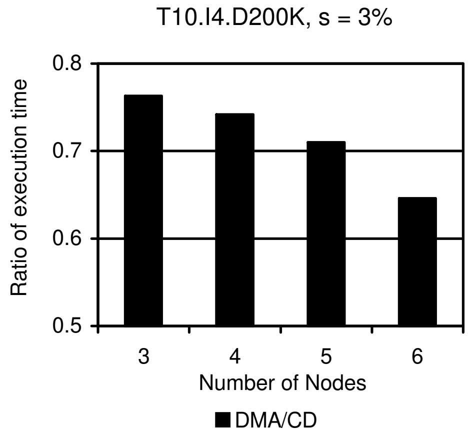

Figure 9: Execution Time (n = 3,4,5,6).

denote the minimum value of the local support counts of all the size $(k-1)$ subsets of X, i.e, $maxsup^i(X) = min\{Y.sup^i \mid Y \subset X \text{ and } |Y| = k-1\}$. It follows from the subset relationship that $maxsup^i(X)$ is an upper bound of the local support count $X.sup^i$. Hence, the sum of these upper bounds over all partitions, denoted by $maxsup(X)$, is an upper bound of $X.sup$. I.e., $X.sup < maxsup(X) = \sum_{i=1}^{n} maxsup^i(X)$. Note that $maxsup(X)$ can be computed at every site at the beginning of the k-th iteration. Since $maxsup(X)$ is an upper bound of its global support count, it can be used for pruning, i.e., if $maxsup(X) < s \times D$, then X cannot be a candidate set. We call this technique *global pruning.* Global pruning can be combined with local pruning to form different pruning strategies. In the following, we outline three possible strategies.

1. *Local Pruning followed by Global Pruning:* After the local pruning, each site S^i can apply global pruning to the remaining candidate sets. The upper bound $maxsup(X)$ for a candidate set X can be fine tuned to
$$X.sup^i + \sum_{j=1, j\neq i}^{n} maxsup^j(X).$$

Since $X.sup^i$ is available during the local pruning, the above upper bound can be computed at site S^i, and it is more effective than the value $maxsup(X)$ in global pruning.

2. *Global Pruning followed by Local Pruning:* Use the upper bound $maxsup(X)$ to prune away some candidate sets at site S^i, and then apply local pruning on the remaining candidate sets. (In the extreme case, we may use global pruning without local pruning).

3. *Global Pruning at Polling Site:* Only local pruning is done at a site during the pruning phase. For a candidate set X, additional pruning is being done at its polling site. Let S^p be the polling site of X and Γ be the set of originating sites from which the requests to do polling on X are being sent. For the sites in Γ, the local support counts of X have been sent to S^p already. For a site S^j not in Γ, since X is not locally large at S^j, the polling site can deduce that its local support count $X.sup^j$ is bounded by the value $min(maxsup^j(X), s \times D^j)$. Therefore an upper bound of $X.sup$ can be computed by

$$\sum_{i \in \Gamma} X.sup^i + \sum_{j=1, j \notin \Gamma}^{n} min(maxsup^j(X), s \times D^j).$$

The above upper bound for X can be used to prune away some candidate sets at a polling site before it starts to collect support counts.

The effectiveness of global pruning depends on the data distribution. For example, let AB be a candidate set and its size-1 subset A is locally large in S^1 but small (not locally large) in S^2, while the size-1 subset B is small in S^1 but large in S^2. By global pruning, it can be deduced that AB is not globally large. On the other hand, if A and B are both large on S^1, and small on S^2, then it cannot be deduced from global pruning that AB is small. In fact, the choice of an appropriate global pruning strategy will depend on the data distribution.

The additional cost in doing global pruning is the storage required to store the local support counts and the message communication to broadcast the support counts. There is a trade-off between the cost and the reduction of candidate sets. It will depend on the data distribution as well as the number of partitions. We believe that global-pruning will pay off when the distribution of the data has certain degree of skewness. Additional performance study is required in order to investigate this technique further.

The hashing technique and relaxation factor proposed in PDM can be integrated with the techniques in DMA [17]. For example, in the selection of hash buckets for broadcasting, the local pruning technique can be used. Also, a relaxation factor on the support threshold can be used to increase the amount of information available at the polling site for global pruning.

Another point worthy to mention here is that the original Count Distribution algorithm as proposed in [5], which is designed for high performance parallel environment, can be improved by introducing polling sites to decrease the amount of message communication required. Its merit is that it requires less synchronization. In fact, in a high performance parallel environment, DMA and CD can be combined to form a hybrid algorithm which has less candidate sets than CD, a slightly more message communication than DMA, but less synchronization. We will investigate this further in our future study.

Another issue related to the performance of the mining of association rules in a distributed database is the difference between the partition sizes. The algorithms such as DMA and CD require some synchronization in each iteration. A large size difference between the partitions would not be favourable to the performance. A possible solution would be to divide some large partitions further to equalize their sizes. This would reduce the time in synchronization. However, the trade-off would be more message communication.

7 Conclusion

We studied an efficient algorithm for mining association rules in distributed databases. The developed method reduces the number of candidate sets at each partition effectively by using local pruning. The communication scheme for count exchange is optimized by using polling sites. The method is implemented and its performance is studied and compared with a direct application of a popular sequential algorithm. The study shows that the proposed technique has superior performance on the mining of association rules in distributed databases.

The efficiency of local pruning can be enhanced by global pruning if local support counts are stored at the sites. We have also discussed the possibility of integrating the techniques in DMA with those in PDM.

Recently, there have been some interesting studies at finding multiple-level or generalized association rules in large transaction databases [13, 21]. An extension of the techniques in DMA to the mining of multiple-level or generalized association rules in distributed database are interesting problems for further research. For experimental purposes, we are planning to implement the DMA and other related algorithms on an IBM SP2 system with 32 nodes to study the problem of mining association rules in a parallel system with high speed communication.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databasess," *Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms*, October 1993.

[2] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami, "An interval classifier for database mining applications," *Proc. 18th Int. Conf. Very Large Data Bases*, pp. 560–573, Vancouver, Canada, August 1992.

[3] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, pp. 914–925, 1993.

[4] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pp. 207–216, May 1993.

[5] R. Agrawal and J. C. Shafer, "Parallel mining of association rules: Design, implementation, and experience," *IBM Research Report RJ1004*, 1996.

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

[7] D.W. Cheung, A. W.-C. Fu, and J. Han, "Knowledge discovery in databases: A rule-based attribute-oriented approach," *Proc. 1994 Int. Symp. on Methodologies for Intelligent Systems*, pp. 164–173, Charlotte, North Carolina, October 1994.

[8] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," *Proc. 1996 IEEE Int. Conf. on Data Engineering*, New Orleans, Louisiana , February 1996.

[9] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1995.

[10] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, "Knowledge discovery in databases: An overview," *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. J. Frawley, editors, pp. 1–27. AAAI/MIT Press, 1991.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R.Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[12] J. Han, Y. Cai, and N. Cercone, "Data-driven discovery of quantitative rules in relational databases," *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, pp. 29–40, 1993.

[13] J. Han and Y. Fu, "Discovery of multiple-level association rules from large databases," *Proc. 1995 Int. Conf. Very Large Data Bases*, Zurich, Switzerland, pp. 420–431, Sept. 1995.

[14] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo, "Finding interesting rules from large sets of discovered association rules," *Proc. 3rd Int. Conf. on Information and Knowledge Management*, pp. 401–408, Gaithersburg, Maryland, Nov. 1994.

[15] R. Ng and J. Han, "Efficient and effective clustering method for spatial data mining," *Proc. 1994 Int. Conf. Very Large Data Bases*, pp. 144–155, Santiago, Chile, September 1994.

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

[17] J.S. Park, M.S. Chen, and P.S. Yu, "Efficient Parallel Data Mining for Association Rules," *Proc. 1995 Int. Conf. on Information and Knowledge Management*, Baltimore, MD, Nov 1995.

[18] G. Piatetsky-Shapiro and W. J. Frawley, *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

[19] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases," *Proc. 1995 Int. Conf. Very Large Data Bases*, pp. 432–444, Zurich, Switzerland, Sept. 1995.

[20] A. Silberschatz, M. Stonebraker, and J. Ullman, "Database Research: Achievements and Opportunities Into the 21st Century," *Report of an NSF Workshop on the Future of Database Systems Research*, May, 1995.

[21] R. Srikant and R. Agrawal, "Mining generalized association rules," *Proc. 1995 Int. Conf. Very Large Data Bases*, pp. 407–419, Zurich, Switzerland, Sept. 1995.

[22] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vols. 1 & 2.* Computer Science Press, 1989.