Communication-Efficient Distributed Mining of Association Rules

Presented at ACM SIGMOD, Boston, May, 2001

Assaf Schuster and Ran Wolff
Computer Science Dept.
Technion - Israel Institute of Technology
{assaf,ranw}@cs.technion.ac.il

ABSTRACT

Mining for associations between items in large transactional databases is a central problem in the field of knowledge discovery. When the database is partitioned among several share-nothing machines, the problem can be addressed using distributed data mining algorithms. One such algorithm, called CD, was proposed by Agrawal and Shafer in [1] and was later enhanced by the FDM algorithm of Cheung, Han et al. [5].

The main problem with these algorithms is that they do not scale well with the number of partitions. They are thus impractical for use in modern distributed environments such as peer-to-peer systems, in which hundreds or thousands of computers may interact. In this paper we present a set of new algorithms that solve the Distributed Association Rule Mining problem using far less communication. In addition to being very efficient, the new algorithms are also extremely robust. Unlike existing algorithms, they continue to be efficient even when the data is skewed or the partition sizes are imbalanced. We present both experimental and theoretical results concerning the behavior of these algorithms and explain how they can be implemented in different settings.

1. INTRODUCTION

1.1 Problem Description

Association Rules Mining (ARM) in large transactional
databases is a central problem in the field of knowledge discovery. The input to the ARM is a database in which objects are grouped by context. An example of such a grouping would be a list of items grouped by the customer who bought them. ARM then requires us to find sets of objects which tend to associate with one other. Given two distinct sets of objects, X and Y, we say Y is associated with X if the appearance of X in a certain context usually implies that Y will appear in that context as well. If X usually implies Y, we then say that the rule $X \Rightarrow Y$ is confident in the database. We would usually not be interested in an association rule unless it appears in more than a certain number of contexts: if it does, we say that the rule has a large support count. The thresholds of support (MinSup) and confidence (MinConf) are parameters of the problem and are usually supplied by the user according to his needs and resources. The solution to the ARM problem is a list of all association rules which have both large support and high confidence in that database. Such lists of rules have many applications in the context of understanding, describing and acting upon the database.

The ARM problem has been investigated intensively during the past few years [2][10][7][8][13][3]. It was shown that the major computational task is the identification of all the large itemsets, those sets of items which have a support count greater than MinSup. Association rules can then be produced from these large itemsets in a fairly straightforward manner. For example, once it is known that both $\{Pasta\,Sauce\}$ and $\{Pasta\,Sauce,\ Parmesan\}$ are large itemsets, the association rule $\{Pasta\,Sauce\} \Rightarrow \{Parmesan\}$ obviously has support, and all that remains is to check if the association is confident.

ARM algorithms are mainly concerned with reducing the number of database scans. This reduction is necessary because the database is usually very large and is stored in secondary memory (disk). The problem was restated in a distributed setting as the Distributed Association Rules Mining (D-ARM) problem. The main reason for restating the problem this way was to parallelize the disk I/O required to solve

it. In D-ARM, the database is partitioned between several parties which can perform independent parallel computations as well as communicate with one another. Several algorithms were proposed to solve D-ARM, most of them for share-nothing machines [1][6][5][11], and some for shared-memory [14] or distributed shared-memory machines [9]. Since the parallelization of disk I/O is itself an easy task, the main show stopper for D-ARM algorithms is communication complexity. The most important factors in the communication complexity of D-ARM algorithms turn out to be the number of partitions, n, and $|C|$, the number of itemsets considered throughout the algorithm.

In this paper we present a set of new algorithms for share-nothing machines. These algorithms improve the communication complexity of the best of the known D-ARM algorithms. We prove that our algorithms are the first to solve the problem with a communication complexity that is linear in n and $|C|$, and with a very small database-dependent multiplicative factor. When compared with current algorithms, our algorithms show an improvement which reaches several orders of magnitude even for mid-range values of n. For higher n values, we prove that the advantage of our algorithms keeps growing at the same rate.

We see possible applications for our communication-efficient algorithms in three main areas. First, they may be used to mine peer-to-peer systems. For example, we may wish to find associations between the mp3 files of different Napster users (more than 1.5 million files in about 10,000 libraries at the time this paper was written). No previous algorithm can cope with $n = 10,000$ with the Internet communication speed available today. Second, we may use these algorithms for broad-scale parallelization of data mining, splitting the problem until each partition fits into the memory of a conventional PC. Third, we can use them in environments where communication bandwidth is an expensive resource, such as billing centers for large communication providers. Although these billing centers usually have fast and wide networks, data mining is performed there as an auxiliary task and the resources it consumes come at the expense of the main system activity.

1.2 Previous Work

The two major approaches to D-ARM as presented in [1] are data distribution (DD) and count distribution (CD). DD focuses on the optimal partitioning of the database in order to maximize parallelism. CD, on the other hand, considers a setting where the data is arbitrarily partitioned horizontally[1] among the parties to begin with, and focuses on parallelizing the computation.

The DD approach is not always applicable. At the time the data is generated, it is often already partitioned. In many cases it cannot be gathered and repartitioned for reasons of security and secrecy, cost of transmission, or sheer efficiency. DD is thus more applicable for systems which are dedicated to performing D-ARM. CD, on the other hand, may be a more appealing solution for systems which are naturally distributed over large expanses, such as stock exchange and credit card systems. This article focuses on the CD approach to D-ARM, although the main ideas can probably be adapted to DD as well.

All the algorithms discussed in this paper are based on the Apriori algorithm [2]. Apriori begins by assuming that any item is a candidate to be a large itemset of size 1. Apriori then performs several rounds of a two phased computation. In the first phase of the k round, the database is scanned and support counts are calculated for all k-sized candidate itemsets (itemsets containing k items). Those candidate itemsets which have support above the user supplied MinSup threshold are considered large itemsets. In the second phase, candidate $k+1$-sized itemsets are generated from the set of large k-sized itemsets if and only if all their k-sized subsets are large. The rounds terminate when the set of large k-sized itemsets is empty.

The CD algorithm [1] is an obvious parallelization of Apriori. In the first phase, each of the parties performs the database scan independently on its own partition. Then a global sum is performed on the support counts of each candidate itemset. Those itemsets whose global support count is larger than MinSup are considered large. The second phase of calculating the candidate $k+1$-sized itemsets can be carried out without any communication because the calculation depends only on the identity of the large k-sized itemsets, which is known to all parties by this time. CD fully parallelizes the disk I/O complexity of Apriori and performs roughly the same computations. CD also requires one synchronization point on each round and carries an $O(|C| \cdot n)$ communication complexity penalty, where C is the group of all candidate itemsets considered by Apriori and n is the number of parties. Since typical values for $|C|$ are tens or hundreds of thousands, CD is obviously not scalable to large numbers of partitions.

In order to reduce the communication load, the CD algorithm was enhanced by FDM [5]. FDM takes advantage of the fact that ARM algorithms only look for rules which are globally large. FDM is based on the inference that in order for an itemset to appear in a certain portion of the transactions in the database it must appear in at least that portion of at least one partition of the database. In FDM, the first stage of

[1] Horizontal partitioning means that each partition includes whole transactions, in contrast with vertical partitioning where the same transaction would be split among several parties.

CD was broken into two rounds of communication. In the first round, every party names those candidate itemsets which are locally large in its partition (e.g. appear in the partition in a frequency greater than or equal to $\frac{MinSup}{|database|}$). In the second round, counts are globally summed for those candidate itemsets which were named by at least one party. If the probability that an itemset will have the potential to be large is $Pr_{potential}$, then FDM only communicates $Pr_{potential} \cdot |C|$ of the itemsets and improves the communication complexity to $O\left(Pr_{potential} \cdot |C| \cdot n\right)$.

The main problem of FDM is that $Pr_{potential}$ is not scalable in n. In 2.2 we show that $Pr_{potential}$ quickly increases to 1 as n increases. The convergence to 1 is especially fast in non-homogeneous databases. Cheung and Xiao presented this problem in [4] and showed that as the non-homogeneity of the database (as measured by the skewness measure they presented) increases, FDM pruning techniques become ineffective. We extend their results by proving that even for moderate non-homogeneity FDM becomes ineffective for a large enough n.

1.3 Our Solution

We first look at the distributive problem of deciding whether an itemset's global support count is above or below a threshold. If we first solve this problem for the set of candidate k-sized itemsets, then support counts of those identified as large can be collected optimally, while support counts for the small itemsets can remain uncollected. We show that solving this distributed agreement problem causes very little overhead.

We use this approach to propose a new family of Apriori-based D-ARM algorithms that improve the communication complexity of current solutions. These algorithms improve data skew robustness and scalability over large numbers of partitions. We show three examples for such algorithms, all of which retain the good time, space and disk I/O complexities of Apriori. Our algorithm for the decision problem has $O\left(Pr_{above} \cdot |C| \cdot n\right)$ communication complexity, where Pr_{above} is the probability that a candidate itemset will have support greater than δ in a specific partition. Pr_{above} is, by definition, smaller than or equal to $Pr_{potential}$. Unlike $Pr_{potential}$, Pr_{above} has no dependency on n and is only dependent on the non-homogeneity of the database. As a result, our algorithms are the first to demonstrate low linear dependency of communication complexity on n and thus, scalability.

After the decision problem was solved, the obvious next step would be to collect the support counts of the large itemsets from all parties with linear output complexity of $O\left(|L| \cdot n\right)$, where L is the group of large itemsets. However, we show that further improvement is possible: the confident rules can be identified with sub-linear communication complexity if another decision problem is solved. This time the decision problem is concerned with the confidence of the rules generated from the large itemsets. By applying our algorithm to this problem too, we demonstrate that it is general and can be implemented for many functions of the database.

The next section describes an algorithm called Distributed Decision Miner (DDM). This algorithm demonstrates our basic approach. We then propose two additional derivatives of DDM, to be described in sections 3 and 4: Preemptive Distributed Decision Miner (PDDM) and Distributed Dual Decision Miner (DDDM). These two algorithms further improve the communication complexity of Distributed Decision Miner. Moreover, the improvements are complementary and we can combine them to get even better results. The combination will not be presented in this context. Section 5 describes in detail the experiments we carried out to verify the algorithm's superiority. We conclude the article in section 6.

1.4 Notations

Let $I = \{i_1, i_2, ..., i_m\}$ be the set of items. A transaction t is a subset of I. Let DB be a list of D such transactions. Let $\overline{DB} = \left\{DB^1, DB^2, ..., DB^n\right\}$ be a partition of DB into n partitions with sizes $\overline{D} = \left\{D^1, D^2, ..., D^n\right\}$ respectively. An itemset is some $X \subseteq I$. Since identical itemsets exist for all parties in all the algorithms in this paper, we will denote them as $X_1, X_2, ..., X_s$. For any X_i and $db \subseteq DB$, let $Support\left(X_i, db\right)$ be the number of transactions in db which contain all the items of X_i. We call $x_i^j = Support\left(X_i, DB^j\right)$ the local support of X_i in partition j and $Support\left(X_i, DB\right)$ its global support.

For some user defined support threshold $0 \leq \delta \leq 1$, we say that the size of X_i is large iff $Support\left(X_i, DB\right) \geq \delta \cdot D$ and the size is small iff $Support\left(X_i, DB\right) < \delta \cdot D$. We say X_i is locally large in the j partition iff $Support\left(X_i, DB^j\right) \geq \delta \cdot D^j$. Let X_a, X_p be two large itemsets such that $X_p \subset X_a$, and let $0 < \lambda \leq 1$ be some user-defined confidence threshold. We say the rule $X_p \Rightarrow X_a \setminus X_p$ is confident iff $Support\left(X_a, DB\right) \geq \lambda \cdot Support\left(X_p, DB\right)$. The D-ARM problem is to distributively find all the rules of the form $X \Rightarrow Y$ while minimizing the amount of communication as well as the number of times $Support\left(\cdot, DB^i\right)$ is evaluated.

The messages the parties send to one another are pairs $\left\langle i, x_i^j \right\rangle$, where i is an itemset number and $x_i^j = Support\left(X_i, DB^j\right)$. We will assume that j, the origin of the message, can be inferred with negligible communication costs. For each party p and itemset X_i, let $G^p\left(X_i\right)$ be the group of all x_i^j such that $\left\langle i, x_i^j \right\rangle$ was received by p. We will assume $G^p\left(X_i\right)$ is equal for all p and refer to it as $G\left(X_i\right)$.

2. THE DISTRIBUTED DECISION MINER ALGORITHM

The basic idea of the Distributed Decision Miner (DDM) algorithm is to verify that an itemset is large before collecting its support counts from all parties. The algorithm differs from FDM in that, in our algorithm, the fact that an itemset is locally large in one partition is not considered sufficient evidence to trigger the collection of all the support counts for that itemset. Instead, the parties perform some kind of negotiation by the end of which they will be able to decide which candidate itemsets are globally large and which are not. The rest is straightforward: the support counts of the large itemsets are collected optimally, with no communication wasted on globally small, but locally large itemsets.

The parties will negotiate by exchanging messages containing local support counts for various itemsets. At any given stage, a common hypothesis H is shared by all parties. This hypothesis concerns the global support of every candidate itemset. Given all the local support counts for an itemset, this hypothesis must correctly predict whether it is large or small. In addition, every party computes another private hypothesis P, based on both the support counts already expressed and the party's local support count for the candidate itemset. For at least one party which has not yet expressed its local support count, and given any subset of the support counts for an itemset, the local hypothesis must correctly predict whether the itemset is large or small.

2.1 The Algorithm

The parties calculate the set of candidate itemsets for which they did not express their local support counts. For each such candidate, every party calculates the global hypothesis H and the local hypothesis P. If H and P disagree whether the size of a candidate itemset is above or below MinSup, then the support count for that candidate should be expressed. The parties express support counts at a certain rate, limited by the bandwidth of the system. Instead of sending one message concerning all the candidates, the parties send smaller messages concerning one or several candidates. No synchronization is required for single messages. Every time a party accepts a message, it updates H and P for the candidate itemsets referred to in that message.

If, for some party, H and P agree on the size of every candidate itemset, that party has nothing to express and it passes on its turn. A party may resume sending messages if arriving messages cause disagreement between H and P about the size of some yet unexpressed candidate itemset. If a full round of passes was received from all parties, then H and P agree for all parties and for every candidate itemset. By the definition of P, there are two possibilities for each candidate itemset: either there is one party whose P correctly predicts the itemset size or all the local support counts have been collected. In the first case, since H and P agree for all parties, and specifically the party with the correct P, then H must be correct for all parties. In the second case, H must be correct by definition.

For Distributed Decision Miner, we define H and P as follows:

$$H(X_i) = \begin{cases} 0 & if \quad G(X_i) = \emptyset \\ \frac{\sum_{x_i^p \in G(X_i)} x_i^p}{\sum_{x_i^p \in G(X_i)} D^p} \cdot D & otherwise \end{cases}$$

$$P\left(X_i, DB^j\right) = \sum_{x_i^p \in G(X_i)} x_i^p + \frac{x_i^j}{D^j} \cdot \sum_{x_i^p \notin G(X_i)} D^p.$$

When estimating H, the parties assume that the unexpressed support counts for each itemset are, on the average, the same as those already expressed. With P, on the other hand, a party assumes that those parties which have not yet expressed their local support counts for that itemset have the same relative support as it does.

As defined, the above assumptions are not required to hold for every party. It is enough that the assumption on H will hold eventually and that the assumption on P holds for one party which has not yet expressed its support count. This is easily proven: Out of all the parties which have not yet expressed support ($x_i^p \notin G(X_i)$), the one with the largest relative support $\frac{x_i^p}{D^p}$ computes a value for P which is an upper bound on the global support count of X_i, and the one with the lowest relative support computes a value for P which is a lower bound on the global support count of X_i. It follows that at least one of those two must always estimate the global support count correctly, thus satisfying the requirement for P. As for H, when all the support counts have been collected, H is equal to that support count. Thus, the requirement from H is also satisfied.

Usually each party can choose which of several candidate itemsets will have its support count sent next. Many heuristics can be used to break ties, but we found one to be most effective. Our tie breaking heuristic is based on the following rationale: whenever two parties are able to express the local support counts of the same candidate itemset, it is best if the one which makes a greater change in P expresses its local support first. If there are opposing parties for a candidate itemset (some of whose P is larger and others whose P is smaller than $\delta \cdot D$), then the one that makes the greater change has the better chance to "convince" opposing parties that they are wrong. If the opposing parties' P is changed to the extent that it now agrees with that of the sending party,

Algorithm 1 Distributed Decision Miner

2.1.0.1 For node j out of n

1. Initialize $C_1 = \{\{i\} : i \in I\}$, $k = 1$, $Passed = \emptyset$
2. While $C_k \neq \emptyset$
   (a) Do
   - Choose an itemset $X_i \in C_k$ which was not yet chosen and for which either $H(X_i) < MinSup \leq P(X_i, DB^j)$ or $P(X_i, DB^j) < MinSup \leq H(X_i)$ and broadcast $\langle i, Support(X_i, DB^j)\rangle$.
   - If no such itemset exists broadcast $\langle pass\rangle$.

   (b) Until $|Passed| = n$
   (c) $L_k = \{X_i \in C_k : H(X_i) \geq MinSup\}$
   (d) Broadcast the support counts for every $X_i \in L_k$ which was never chosen.
   (e) $C_{k+1} = Apriori\_Gen(L_k)$
   (f) $k = k + 1$
3. $Gen\_Rules(L_1, L_2, ..., L_k)$

2.1.0.2 When node j receives a message M from node p:

1. If $M = \langle pass\rangle$ insert p into $Passed$
2. Else if $|Passed| = n$ then M is the support counts of itemsets p has not yet sent. Update accordingly.
3. Else $M = \langle i, Support(X_i, DB^p)\rangle$
   - If $p \in Passed$ then remove p from $Passed$
   - Recalculate $H(X_i)$ and $P(X_i, DB^p)$

the opposing parties will refrain from expressing their own support and thus save the cost of messages. It is therefore a good strategy for a party to send those support counts which will cause the greatest change in the Ps of opposing parties, for those same itemsets.

When party k expresses support for itemset X_i, the influence on P of party l is equal to $\left|x_i^k - \frac{x_i^l}{D^l} \cdot D^k\right|$. However, since x_i^l has not yet been expressed , we estimate the change as $R(X_i, DB^k) = \left|x_i^k - \frac{H(X_i)}{D} \cdot D^k\right|$. We use the rating given by $R(X_i, DB^j)$ for breaking the tie. The tie is broken by selecting the itemsets with the greatest rating.

2.2 Complexity Analysis

The messages sent by Distributed Decision Miner can be separated into two classes: messages relating to itemsets which eventually turn out to be large and messages relating to itemsets which eventually turn out to be small. While the messages related to large itemsets are needed anyway for the calculation of confidence, those relating to small itemsets are wasted. The wasted messages can also be separated into two classes: those which imply that the itemset being considered is large (strengthening evidence) and those which imply that the itemset is small (weakening evidence).

By definition, every message containing strengthening evidence must be presented by a party with relative local support $\frac{Support(X_i, DB^j)}{D^j} \geq \delta$. Thus, the expected number of such messages is $O(Pr_{above} \cdot |C| \cdot n)$, where n is the number of parties, C is the group of all itemsets considered by Distributed Decision Miner (which is equal to that considered by Apriori and FDM), and Pr_{above} is the probability that a specific itemset is locally large in a specific partition.

For a certain small itemset, let the number of wasted messages containing strengthening evidence be s and the number of wasted messages containing weakening evidence be w. Let their average distances from δ be ϵ^s and ϵ^w accordingly. In the worst case, all the possible strengthening evidence for that small itemset is collected. If we remove the last message containing weakening evidence, we can be sure that $H(X_i) \geq \delta \cdot D$; otherwise, the message containing it would never have been sent. We assume, for simplicity, that the partitions have the same size and that the final weakening message was of average distance from δ. We then have $\delta \cdot (s + w - 1) < s \cdot (\delta + \epsilon^s) + (w-1) \cdot (\delta - \epsilon^w)$. It follows that $w < s \cdot \frac{\epsilon^s}{\epsilon^w}$. ϵ^s is only dependent on the distribution and not on C or n (the variability of ϵ^s has negative dependency on n). ϵ^w can only increase with n because the algorithm has more possible messages with weakening evidence to choose from and it tends to choose those with more extreme values. Thus, the number of such messages has linear

or lower dependency on the number of messages containing strengthening evidence, and the total communication complexity is $O\left(Pr_{above} \cdot |C| \cdot n\right)$.

By comparison, the communication complexity of FDM is $O\left(Pr_{potential} \cdot |C| \cdot n\right)$, where $Pr_{potential}$ is the probability that any of the partitions has relative support larger than δ. If we assume that the support counts of a small itemset in different partitions are independent, then $Pr_{potential} = 1 - (1 - Pr_{above})^n$. This converges to 1 very quickly, even for a small Pr_{above}. It follows that for a large number of partitions, FDM performs as poorly as CD in terms of communication.

It is clear that Distributed Decision Miner improves considerably over FDM with regard to three factors: a decrease of Pr_{above}, an increase of the ratio of small vs. large itemsets, and an increase of n. These three factors behave quite differently. Pr_{above} is the most influential of the three. As can be seen in the above analysis, DDM can be as much as $\frac{1}{Pr_{above}}$ better than FDM. However, it should be noted that under the the CD approach to D-ARM, Pr_{above} is a property of the database and cannot be controlled by the algorithm. In the databases we checked, Pr_{above} had values of between 0.05 to 0.5. The ratio between the number of small and large itemsets is important for Distributed Decision Miner because of the following extreme case: if all the itemsets are large, quick identification them is of no benefit, and we would have to collect all counts anyway. Nevertheless, in section 4 we will present an algorithm that can still give considerable improvement even in this case. Finally, n has a very large effect for small values of n. Then, as n gets larger, its effect becomes linear.

The main extra computational task of Distributed Decision Miner is the selection of an itemset to be sent. The fastest way to do this is by managing a priority queue with the priority of a potential itemset being $R\left(X_i, DB^j\right)$, and updating it every time a message is received. Every such update requires $O\left(\log\left(|C|\right)\right)$ time, so the time complexity of FDM is multiplied in Distributed Decision Miner by $\log\left(|C|\right)$, which is not significantly larger. Space complexity (disregarding the database storage, which is of course the same size) is n times larger than that of FDM because of the need to store, for each itemset, certain attributes of the other parties.

3. PREEMPTIVE DISTRIBUTED DECISION MINER

In the course of distributed mining, we often discover that some partition contains the most convincing evidence for the sizes of itemsets. For example, a large partition can sometimes, by itself, have a support count for a given itemset that is greater than $\delta \cdot D$. On the other hand, if such a large partition has no support count for an itemset, it can influence P so strongly that other parties will be convinced not to express their own support counts. When the database is strongly skewed, such phenomena are especially abundant.

We would like to allow parties which have more convincing evidence (extreme support counts) to send their support counts at an earlier stage of the negotiation in the hope that their evidence will shorten negotiation time and reduce communication. Similarly, we would like parties which do not have convincing evidence to refrain from sending messages so as not to use bandwidth which can be better employed. As we showed in the previous section, the rating function $R\left(X_i, DB^k\right) = \left|x_i^k - \frac{H(X_i)}{D} \cdot D^k\right|$ gives the k^{th} party an estimation for the effectiveness of each of its possible messages. We would like the series of messages to have a constantly decreasing value of R. In this section we show that parties can coordinate their messages and come nearer to this goal if each of them weighs the importance of its information against that of information contributed by other parties.

3.1 The Algorithm

In an R-optimal negotiation, the received messages have decreasing R values. Generating such a series requires, however, global knowledge, which is not available to the parties. We achieve a near monotonously decreasing series of R values by selecting as a leader the party which sent the message with the maximal R. We keep a record of the leader's identity and the R value of its last message. We do not allow any other party to send messages unless the R of its message is greater than that of the last message sent by the leader. If some other party sends a message with an R greater than that of the leader, this party then replaces the leader.

Preventing other parties from sending messages does not effect the correctness of the algorithm because the algorithm still terminates in the same state. We must make sure, however, that a leader passes the leadership to another party when it decides to pass on its turn; otherwise, the algorithm might not terminate. We do this by setting the value of the leader's last R to zero when the leader passes on its turn. When the leader's last R is zero, any party that has any message to send will send it and a party that has no message to send will pass on its turn. It is easy to calculate the leader's R for the R proposed in the prior section. For other R functions though, the calculation may be difficult or even impossible (for example, it is impossible if R is random).

It is important to note that R can be extended to include other properties of the sent message. For example, R can be used to encode information about the cost of sending the message, whether that be in time (e.g. smaller bandwidth for that party) or in money

(if this message is sent, for example, over a WAP channel). Preemptive Distributed Decision Miner will try to reach an R-optimal negotiation regardless of what R encodes.

3.2 Complexity Analysis

The complexity of Preemptive Distributed Decision Miner is dependent on the skewness of the database. In extreme cases where only one partition is significant, the communication can decrease to $O\left(Pr_{above} \cdot |C|\right)$ because only the party with that partition ever sends messages. On the other hand, when the partitions are very homogeneous, the parties may constantly compete over leadership. In this case, Preemptive Distributed Decision Miner reverts to Distributed Decision Miner, with communication complexity of $O\left(Pr_{above} \cdot |C| \cdot n\right)$. Note that Preemptive Distributed Decision Miner's objective is to be R-optimal. If R does not provide a good measurement of the significance of potential messages, Preemptive Distributed Decision Miner may perform even more poorly than Distributed Decision Miner.

If the time required for a party which receives a message to compute R of that message is $O\left(1\right)$, Preemptive Distributed Decision Miner will have the same time and space complexities as Distributed Decision Miner.

4. DISTRIBUTED DUAL DECISION MINER

In their groundbreaking article [2], Agrawal and Srikant gave the following definition of ARM:

> "Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively."

Until now all the known algorithms actually gave, in addition to the list of rules, their respective support counts and confidence. As we will show here, in the distributed setting, we can detect whether rule support count and confidence are larger or smaller than the required minimum without ever fully calculating them.

The basic idea is that we can use a DDM-type algorithm to detect all the large itemsets very efficiently. We need to collect the global support counts of the large itemsets inorder to verify that the confidence of the rule is above the given confidence threshold. However, we must remember that our goal is only to decide if the confidence of a rule is above or below a given threshold, and not to find the exact confidence of the rule. This is exactly the same distinction we

Algorithm 2 Preemptive Distributed Decision Miner

For node j out of n

1. Initialize $C_1 = \{\{i\} : i \in I\}$, $k = 1$, $Passed = \emptyset$, $leader = j$, $last\_R = 0$
2. While $C_k \neq \emptyset$
   (a) Do
   - Choose an itemset $X_i \in C_k$ which was not yet chosen and for which either $H\left(X_i\right) < MinSup \leq P\left(X_i, DB^j\right)$ or $P\left(X_i, DB^j\right) < MinSup \leq H\left(X_i\right)$ and which maximizes $R\left(X_i, DB^j\right)$.
   - If such itemset exists and either $R\left(X_i, DB^j\right) > last\_R$ or $leader = j$ broadcast $\left\langle i, Support\left(X_i, DB^j\right)\right\rangle$
   - Else broadcast $\langle pass \rangle$.

   (b) Until $|Passed| = n$
   (c) $L_k = \{X_i \in C_k : H\left(X_i\right) \geq MinSup\}$
   (d) Broadcast the support counts for every $X_i \in L_k$ which was never chosen.
   (e) $C_{k+1} = Apriori\_Gen\left(L_k\right)$
   (f) $k = k + 1$
3. $Gen\_Rules\left(L_1, L_2, ..., L_k\right)$

3.1.0.3 *When node j receives a message M from node p:*

1. If $M = \langle pass \rangle$ insert p into $Passed$
2. Else if $|Passed| = n$ then M is the support counts of itemsets p has not yet sent. Update accordingly.
3. Else $M = \langle i, Support\left(X_i, DB^p\right)\rangle$
   - If $p \in Passed$ then remove p from $Passed$
   - Recalculate $H\left(X_i\right)$ and $P\left(X_i, DB^p\right)$
   - If $leader = p$ then update $last\_R = R\left(X_i, DB^p\right)$
   - Else if $last\_R < R\left(X_i, DB^p\right)$
     - Update $last\_R = R\left(X_i, DB^p\right)$
     - Update $leader = p$

made when we presented Distributed Decision Miner. As we will show here, this second distributed decision problem can be solved in a similar manner.

To illustrate this idea, we present the following two examples:

1. Assume that the rule $Pasta\,Sauce \Rightarrow Parmesan$ is possible in the database. Assume also that it is locally large in every partition, but confident in none (e.g. for every p $Support\left(Parmesan \wedge Pasta\,Sauce, DB^p\right) > \delta \cdot |DB^p|$, $Support\left(Pasta\,Sauce, DB^p\right) \geq \delta \cdot |DB^p|$, $\frac{Support(Parmesan \wedge Pasta\,Sauce, DB^p)}{Support(Pasta\,Sauce, DB^p)} < \lambda$). Using Distributed Decision Miner, three messages are required to identify that both $Parmesan \wedge Pasta\,Sauce$ and $Pasta\,Sauce$ are significant (compared to $6 \cdot n$ in FDM). With the algorithms described in sections 2 and 3, we would need additional $2 \cdot (n-1)$ messages to collect the local support counts of the remaining parties for $Pasta \wedge Pasta\,Sauce$ and $Pasta\,Sauce$ before we could judge if $Pasta\,Sauce \Rightarrow Parmesan$ is significant. However, note that if for no party the local confidence is above λ, then the global confidence cannot be above λ. By implementing an algorithm similar to FDM we could have pruned this rule without sending a single message.
2. Assume that this same rule is both supported and confident in every partition. If one party suggests that the rule is globally confident and no other party objects, this is enough to determine that the rule is indeed globally significant.

In Distributed Dual Decision Miner (DDDM) we will generalize these two examples: first, by defining H and P for rules as well as for itemsets, and then by performing a negotiation similar to the one we performed for the support count to decide whether or not potential rules are confident.

4.1 The Algorithm

Distributed Dual Decision Miner runs any Distributed Decision Miner variant[2] to identify large itemsets without performing stage 2.(d), the collection of yet uncollected support counts. Then, instead of calling the original *Gen_Rules* procedure, it uses another variant of DDM called Distributed Decision Confidence Miner to mine the set of rules with confidence above a user-defined threshold λ.

We first describe a simple algorithm which mines rules with large confidence and then, in subsection 4.3, introduce a rule pruning method which does away with the need to consider many of the rules. Distributed

[2]We use here, for reasons of clarity, Distributed Decision Miner. But Preemptive Distributed Decision Miner can be used as well.

Algorithm 3 Distributed Dual Decision Miner

4.1.0.4 *For node j out of n*

1. Initialize $C_1 = \{\{i\} : i \in I\}$, $k = 1$, $Passed = \emptyset$
2. While $C_k \neq \emptyset$
   (a) Do
   - Choose an itemset $X_i \in C_k$ which was not yet chosen and for which either $H(X_i) < MinSup \leq P\left(X_i, DB^j\right)$ or $P\left(X_i, DB^j\right) < MinSup \leq H(X_i)$ and broadcast $\left\langle i, Support\left(X_i, DB^j\right)\right\rangle$.
   - If no such itemset exists broadcast $\langle pass \rangle$.

   (b) Until $|Passed| = n$
   (c) $L_k = \{X_i \in C_k : H(X_i) \geq MinSup\}$
   (d) $C_{k+1} = Apriori\_Gen(L_k)$
   (e) $k = k + 1$
3. $Mine\_Rules(L_1, L_2, ..., L_k)$

4.1.0.5 *When node j receives a message M from node p:*

1. If $M = \langle pass \rangle$ insert p into $Passed$
2. Else if $|Passed| = n$ then M is the support counts of itemsets p has not yet sent. Update accordingly.
3. Else $M = \langle i, Support(X_i, DB^p) \rangle$
   - If $p \in Passed$ then remove p from $Passed$
   - Recalculate $H(X_i)$ and $P(X_i, DB^p)$

Decision Confidence Miner makes one round of negotiations to decide which of the candidate rules $X_p \Rightarrow X_a \backslash X_p$ have $Support\left(X_a, DB\right) \geq \lambda \cdot Support\left(X_p, DB\right)$. The algorithm sends messages of three types, $\left\langle rule\_id, x_p^i, x_a^i \right\rangle$, $\left\langle rule\_id, x_p^i \right\rangle$ or $\left\langle rule\_id, x_a^i \right\rangle$, depending on which of the two support counts was expressed, where $rule\_id$ is the number of the rule in some deterministic enumeration and $x_j^i = Support\left(X_j, DB^i\right)$. Again we define H and P with the same limitations as they had in previous algorithms.

The functions we define in Distributed Decision Confidence Miner are[3]:

$$G\left(X_p \Rightarrow X_a \setminus X_p\right) = \left\{ j : x_p^j \in G\left(X_p\right) \wedge x_a^j \in G\left(X_a\right) \right\}$$

$$P\left(X_p \Rightarrow X_a \setminus X_p, DB^i\right) =$$

$$\frac{\sum_{l \in G\left(X_p \Rightarrow X_a \setminus X_p\right)} x_a^l + \left(n - \left|G\left(X_p \Rightarrow X_a \setminus X_p\right)\right|\right) \cdot x_a^i}{\sum_{l \in G\left(X_p \Rightarrow X_a \setminus X_p\right)} x_p^l + \left(n - \left|G\left(X_p \Rightarrow X_a \setminus X_p\right)\right|\right) \cdot x_p^i}$$

$$H\left(X_p \Rightarrow X_a \setminus X_p\right) =$$

$$\begin{cases} \frac{\sum_{l \in G\left(X_p \Rightarrow X_a \setminus X_p\right)} x_a^l}{\sum_{l \in G\left(X_p \Rightarrow X_a \setminus X_p\right)} x_p^l} & if \quad \left|G\left(X_p \Rightarrow X_a \setminus X_p\right)\right| > 0 \\ 0 & otherwise \end{cases}$$

$$R\left(X_p \Rightarrow X_a \setminus X_p, DB^i\right) = \left| \frac{x_a^i - \frac{H\left(X_a\right)}{D} \cdot D^i}{x_p^i - \frac{H\left(X_p\right)}{D} \cdot D^i} \right|$$

To prove that Distributed Decision Confidence Miner works, it is enough to show that H and P have the properties defined in 2.1. For one party which has not yet expressed both parts of a rule, P is required to be an upper bound on the global confidence, and for another it is required to be a lower bound. The requirement for H is that H is correct if all the data was gathered. Clearly, when considering the parties not in $G\left(r_l\right)$, the one with the maximal $\frac{x_a^i}{x_p^i}$ computes a value for P which is an upper bound on the confidence, and the one with the minimal $\frac{x_a^i}{x_p^i}$ computes a value for P which is a lower bound. Thus, the requirement from P is satisfied. In addition, when all the support counts of a rule are collected, H must be correct because it is equal to the confidence. Thus, the requirement for H is met.

4.2 Complexity Analysis

Distributed Decision Miner and Preemptive Distributed Decision Miner reduce communication complexity by

[3]Note that we use here both G of a rule and G of an itemset.

Algorithm 4 Distributed Decision Confidence Miner

4.1.0.6 *For node i of n nodes*

1. Initialize
   - R_l to be the set of all rules $X_p \Rightarrow X_a \setminus X_p$ such that $X_p, X_a \in L$ and $X_p \subset X_a$.
   - $Passed = \emptyset$.
2. Do
   - Choose r_k to be some $X_p \Rightarrow X_a \setminus X_p \in R_l$ such that $i \neq G\left(r_k\right)$ and either $H\left(r_k\right) < \lambda \leq P\left(r_k, DB^i\right)$ or $P\left(r_k, DB^i\right) < \lambda \leq H\left(r_k\right)$ and broadcast $\left\langle k, Support\left(X_p, DB^i\right), Support\left(X_a, DB^i\right)\right\rangle$.
   - If there is no such r_k broadcast $\langle pass \rangle$.
3. Until $|Passed| = n$
4. $R = \left\{ r_k \in R_l : H\left(r_k\right) \geq \lambda \right\}$

4.1.0.7 *When node i receives a message M from node j:*

1. If $M = \langle pass \rangle$ insert p to $Passed$.
2. Else $M = \left\langle k, Support\left(X_p, DB^j\right), Support\left(X_a, DB^j\right)\right\rangle$
   - If $i \in Passed$ remove i from $Passed$.
   - Recalculate $G\left(r_l\right)$ for every r_l which includes X_a and/or X_p, if $G\left(r_l\right)$ changes update $H\left(r_l\right)$ and $P\left(r_l\right)$ as well.

reducing the communication required for the identification of small itemsets. Distributed Dual Decision Miner, in contrast, reduces the communication by refraining from collecting local counts of large itemsets.

When discussing the theoretical bounds of Distributed Dual Decision Miner, one must note that the distribution of large itemsets over partitions in a real database may have a very special form. Simplistic assumptions about that distribution, which were acceptable for small itemsets, may mislead us with regard to the actual savings in communication. As a result, we will say only that in the algorithm's worst case, all the support counts of all itemsets are collected in the itemset identification stage. In this case, Distributed Decision Confidence Miner is trivial and requires no communication at all. Thus, the algorithm's performance is exactly the same as that of Distributed Decision Miner. Further analysis will be done at the experimental level only.

4.3 Rule Pruning

The number of rules which can be generated from a given set of large itemsets is enormous. If we check all the potential rules induced by a single k sized large itemset X we would have to check every rule $X_p \Rightarrow X_a \setminus X_p : X_p \subset X_a \subseteq X$. This is a total of $\sum_{i=1}^{k} \left(\begin{array}{c} k \\ i \end{array} \right) \sum_{j=1}^{i} \left(\begin{array}{c} i \\ j \end{array} \right) = 3^k$ potential rules. We use the following observation to prune rules: If X_p, X_a are two itemsets, such that $X_p \subset X_a$ and the confidence of $X_p \Rightarrow X_a \setminus X_p$ is below the $MinConf$ threshold, then for any $X_{pp} \subset X_p$, the confidence of $X_{pp} \Rightarrow X_a \setminus X_{pp}$ is also below $MinConf$. Similarly, for any $X_{aa} \supset X_a$, the confidence of $X_p \Rightarrow X_{aa} \setminus X_p$ is below $MinConf$. This observation is correct because $Support\left(X_p, DB\right) \leq Support\left(X_p, DB\right)$ and $Support\left(X_{aa}, DB\right) \leq Support\left(X_a, DB\right)$. If, on the other hand, the rule $X_p \Rightarrow X_a \setminus X_p$ is confident, then for every $X_p \subset X_{pp} \subset X_{aa} \subseteq X_a$, the rule $X_{pp} \Rightarrow X_{aa} \setminus X_{pp}$ is confident as well.

This observation allows us to alter Distributed Decision Confidence Miner by splitting it into several rounds. At each round, many of the possible rules can either be pruned or inferred with no communication. We initialize the candidate rule set R_k with a single rule $R_0 = \{\emptyset \Rightarrow \emptyset\}$, which must be both supported and confident. In each round we run an algorithm similar to Distributed Decision Confidence Miner to decide which of the rules in R_k are confident. We develop some new candidate rules according to the following two candidate generation methods: If a rule is found to be confident than every rule which specify the precendent or generalize the antecent of that rule must also confident and every rule which further specify the antecendent is considered a candidate. If, on the other hand, a rule was found not to be confident than any rule which specify the precendent is still a candidate.

Algorithm 5 Pruning Distributed Decision Confidence Miner

4.3.0.8 *Definitions:*

$large\_extentions\left(X_j\right) = \{X_k \in L : X_k = X_j \cup \{i\} \ for\ some\ i \in I\}$

4.3.0.9 *For node i of n nodes:*

1. Initialize: $R_0 = \{\emptyset \Rightarrow \emptyset\}$, $k = 0$, $R = \emptyset$
2. While $R_k \neq \emptyset$
   (a) Initialize $Passed = \emptyset$
   (b) Do
   - Choose r_l to be some $X_p \Rightarrow X_a \setminus X_p \in R_k$ such that $i \neq G\left(r_l\right)$ and either $H\left(r_l\right) < \lambda \leq P\left(r_l, DB^i\right)$ or $P\left(r_l, DB^i\right) < \lambda \leq H\left(r_l\right)$ and broadcast $\left\langle l, Support\left(X_p, DB^i\right), Support\left(X_a, DB^i\right)\right\rangle$.
   - If there is no such r_l broadcast $\langle pass \rangle$.

   (c) Until $|Passed| = n$
   (d) For each $r_l = X_p \Rightarrow X_a \setminus X_p \in R_k$ such that $H\left(r_l\right) < MinConf$ $R_{k+1} = R_{k+1} \cup \{X_{pp} \Rightarrow X_a \setminus X_{pp} : X_{pp} \in large\_extensions\left(X_p\right)\}$
   (e) For each $r_l = X_p \Rightarrow X_a \setminus X_p \in R_k$ such that $H\left(r_l\right) \geq MinConf$
   - $R_{k+1} = R_{k+1} \cup \{X_p \Rightarrow X_{aa} \setminus X_p : X_{aa} \in large\_extensions\left(X_a\right)\}$
   - $R = R \cup \{X_{pp} \Rightarrow X_{aa} \setminus X_{pp} : X_p \subseteq X_{pp} \subset X_{aa} \subseteq X_a\}$

   (f) $k = k + 1$

4.3.0.10 *When node i receives a message M from node j:*

1. If $M = \langle pass \rangle$ insert p to $Passed$.
2. Else $M = \left\langle k, Support\left(X_p, DB^j\right), Support\left(X_a, DB^j\right)\right\rangle$
   - If $i \in Passed$ remove i from $Passed$.
   - Recalculate $G\left(r_l\right)$ for every r_l which includes X_a and/or X_p, if $G\left(r_l\right)$ changes update $H\left(r_l\right)$ and $P\left(r_l\right)$ as well.

5. EXPERIMENTAL RESULTS

We used synthetic databases generated with the gen tool (available from Quest web site at [12]), which is based on ideas published in [2]. We generated several large databases and then sampled them to generate the partitioned database. The sample sizes were kept to less than 10% of the original database in order to reduce the dependency between different experiments. We systematically scanned MinSup values of 3% to 30% of the size of the database in order to sidestep the risks of parameter tuning. We used the reasonable $MinConf = 0.5$. Communication load was measured assuming 4 bytes for support count encoding and 2 bytes for itemset number encoding.

Figure 1, 2 and 3 show the typical communication loads of CD, FDM, Distributed Decision Miner, Preemptive Distributed Decision Miner and Distributed Dual Decision Miner on various unskewed databases. It can be seen that the latter three algorithms uses far less communications than the former two. It can also be seen that both CD and FDM are non-scalable with respect to either n or MinSup. For unskewed databases, PDDM and DDM behavior is effectively the same, and DDDM is up to 30% more efficient once the amount of communication is large enough.

6. CONCLUSIONS

During the past few years, distributed systems have become a mainstream computing paradigm. Whether it be a company's Virtual Private Network, a multi-server billing center, a network of independent stock-brokers or a peer-to-peer mp3 library like Napster, the wealth of information available on-line is constantly expanding. This information is by and large distributed, and there is a growing need for tools that will assist in understanding and describing it.

These new databases differ from distributed databases of the past. The partitioning of the data is usually skewed. The connections between partitions are sparse and often unreliable, and various throughputs and latencies may apply. Distributed knowledge discovery algorithms will, in our view, become a major tool in the making, maintaining and analysis of distributed systems. This will require us to change our approach to distributed knowledge discovery and accept the skewed, sparsely connected, sometimes unreliable environment of these distributed systems. We will have to restate well-known problems and define new ones.

This paper can be viewed as an example of a new approach to one such well-known problem. D-ARM problem was restated here as a decision problem, negotiated among different parties. The resulting algorithms are both more efficient, more resiliant to data skewness, and better able to overcome certain communication difficulties such as unordered messages, variable or uneven throughputs and the like. We intend to further extend the concept of mining through distributed decision-making and apply it to other areas of knowledge discovery.

7. REFERENCES

[1] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962 – 969, 1996.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases (VLDB94)*, pages 487 – 499, Santiago, Chile, September 1994.

[3] R. Bayardo and R. Agrawal. Mining the most interesting rules. In *Proc. of the ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 145 – 154, San Diego, CA USA, 1999.

[4] D. Cheung and Y. Xiao. Effect of data skewness in parallel mining of association rules. In *12nd Pacific-Asia Conference on Knowledge Discovery and Data Mining.*, pages 48 – 60, April 1998.

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

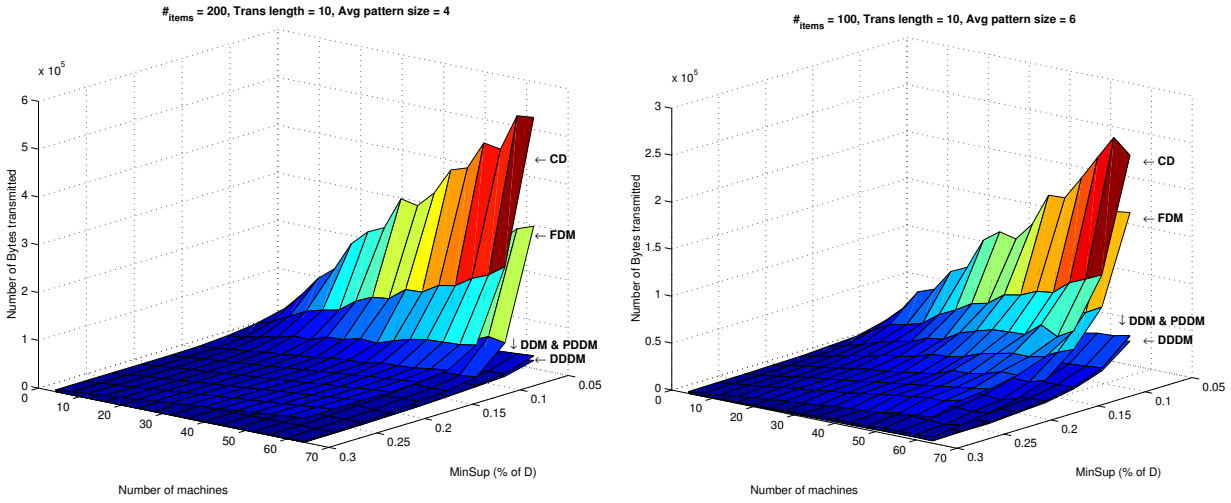
[6] Eui-Hong (Sam) Han, George Karypis, and Vipin Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):352 – 377, 2000.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. Technical Report 99-12, Simon Fraser University, October 1999.

[8] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *Proc. of the 21st International Conference on Very Large Data Bases (VLDB95)*, pages 420 – 431, Zurich, Switzerland, September 1995.

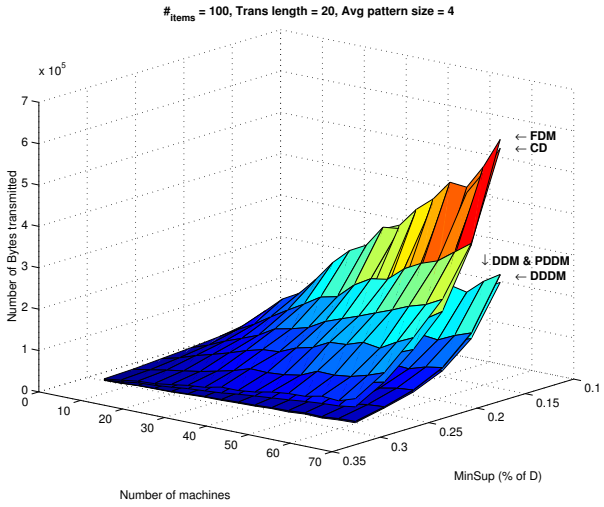
[9] Zoltan Jarai, Aashu Virmani, and Liviu Iftode. Towards a cost-effective parallel data mining approach. In *Workshop on High Performance Data Mining, (held in conjunction with IPPS'98)*, March 1998.

[10] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD95)*, pages 175 – 186, San Jose, California, May 1995.

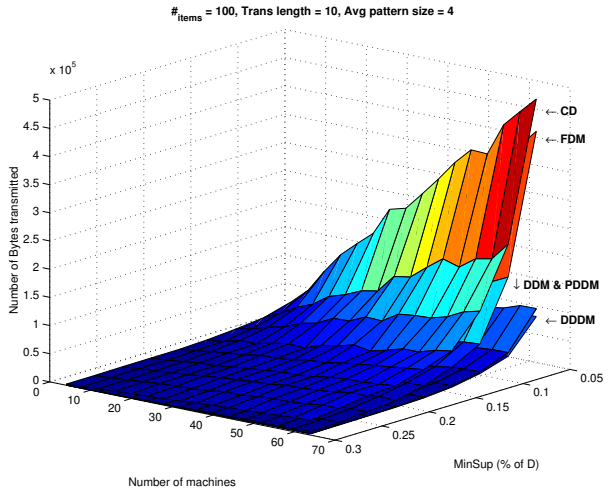


(a)

(b)

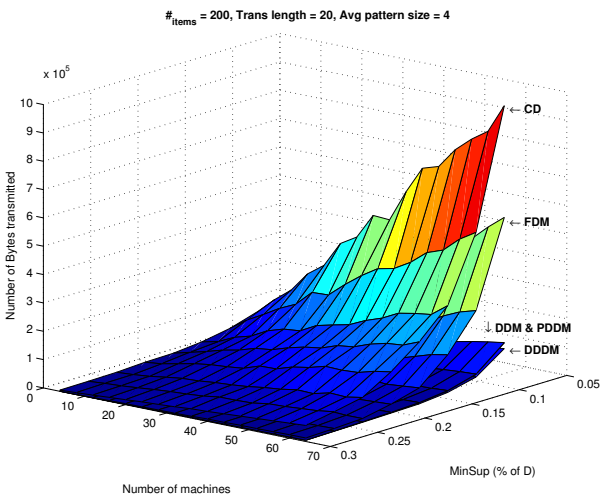


(c)

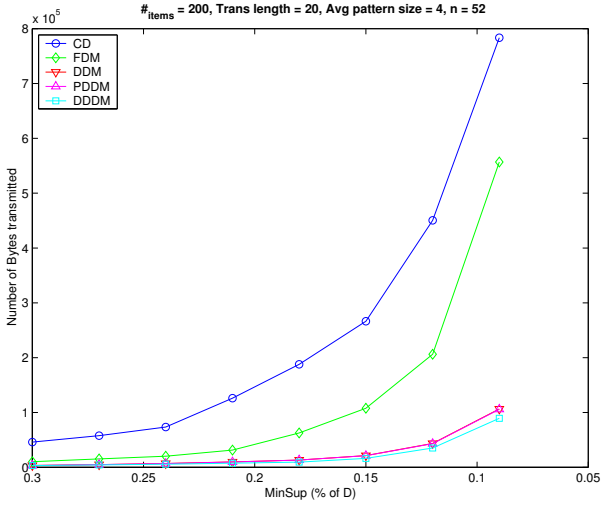


(d)

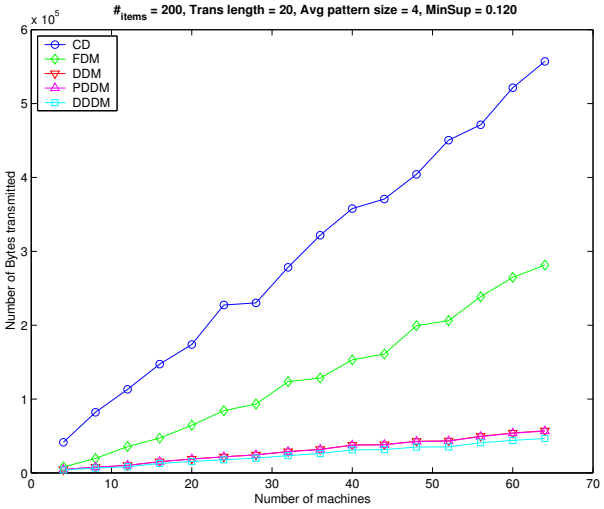
Figure 1: figures 1(a), 1(b), 1(c) and 1(d) show the typical performance of CD, FDM, DDM, PDDM and DDDM over several unskewed databases, measured by the number of transmitted bytes vs. n and MinSup. The MinSup parameter is related to both the number of candidates and Pr_{above}. It can be seen that CD and FDM are not scalable with neither n nor MinSup. Note that when the partition is unskewed, DDM and PDDM are effectively the same algorithm because there are no obvious leaders.



(a)



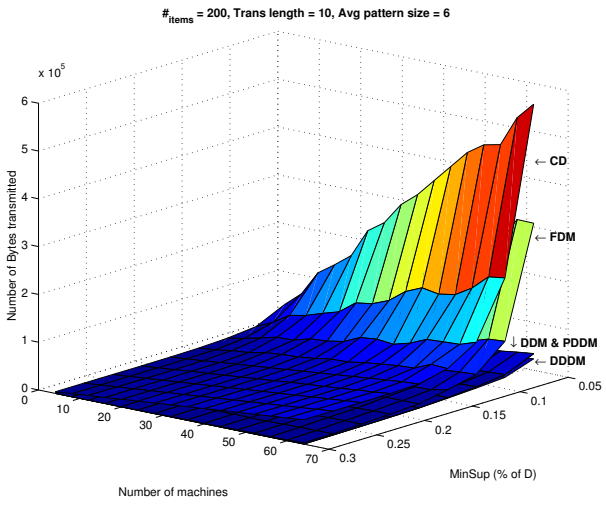
(b)



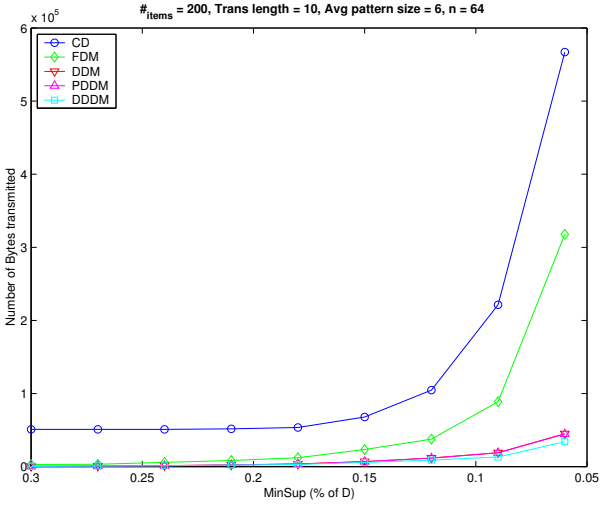
(c)

	16		52	
0.12	CD	147200	CD	450528
	FDM	47190	FDM	206160
	DDM	15360	DDM	43480
	$PDDM$	15244	$PDDM$	43184
	$DDDM$	12892	$DDDM$	35270
0.30	CD	14144	CD	45968
	FDM	2576	FDM	10296
	DDM	1070	DDM	3386
	$PDDM$	1056	$PDDM$	3292
	$DDDM$	1062	$DDDM$	3186

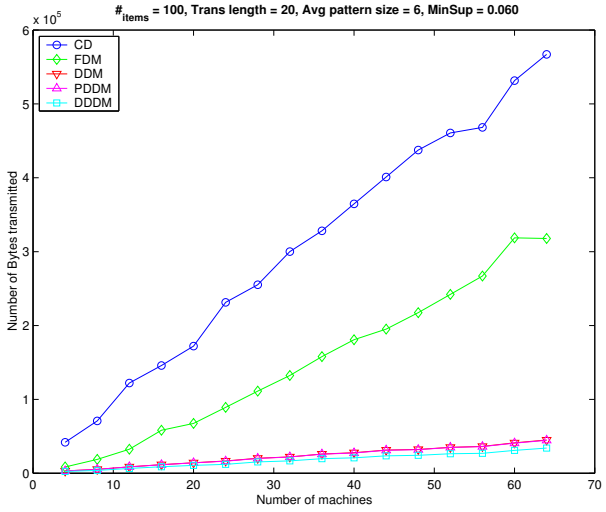
Figure 2: Figure 2(a) showes the performance of CD, FDM, DDM, PDDM and DDDM on an unskewed database like those of 1(a) through 1(d). Here we added views of the performance for fixed n (2(b)) and fixed MinSup (2(c)) and a sample of values for all algorithms for $n = 16, 52$ and $Minsup = 0.12 \cdot D, 0.3 \cdot D$. It can be seen from 2(b) and 2(c) that once the amount of communication is large enough, DDDM performs better than DDM by $10 - 30$ percent.



(a)



(b)



(c)

	8		56	
0.06	CD	70880	CD	468160
	FDM	18756	FDM	266916
	DDM	5242	DDM	36340
	$PDDM$	5286	$PDDM$	36058
	$DDDM$	3816	$DDDM$	26918
0.30	CD	6368	CD	44576
	FDM	218	FDM	1650
	DDM	46	DDM	312
	$PDDM$	42	$PDDM$	252
	$DDDM$	36	$DDDM$	270

Figure 3: Figure 3(a) shows the performance of CD, FDM, DDM, PDDM and DDDM on yet another unskewed database. When the amount of communication increases, DDDM performs better than DDM by 30%. The performance of DDM alone is ~ 3 orders of magnitude better than FDM.

[11] John Shafer, Rakesh Agrawal, and Manish Mehta. Sprint. a scalable parallel classifier for data mining. In *Proc. of the 22nd Intl Conference on Very Large Databases*, pages 544 – 555, Bombay, India, September 1996.

[12] R. Srikant. Synthetic data generation code for association and sequential patterns. Via Web, at http://www.almaden.ibm.com/cs/quest/.

[13] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases (VLDB94)*, pages 407 – 419, Santiago, Chile, September 1994.

[14] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processors. In *Proc. of Supercomputing96*, pages 17 – 22, Pittsburg, PA, November 1996.