A High-Performance Distributed Algorithm for Mining Association Rules*

Assaf Schuster, Ran Wolff, and Dan Trock
Technion – Israel Institute of Technology
Email: {assaf,ranw,dtrock}@cs.technion.ac.il

Abstract

We present a new distributed association rule mining (D-ARM) algorithm that demonstrates superlinear speedup with the number of computing nodes. The algorithm is the first D-ARM algorithm to perform a single scan over the database. As such, its performance is unmatched by any previous algorithm. Scale-up experiments over standard synthetic benchmarks demonstrate stable run time regardless of the number of computers. Theoretical analysis reveals a tighter bound on error probability than the one shown in the corresponding sequential algorithm.

1 Introduction

The economic value of data mining is today well established. Most large organizations regularly practice data mining techniques. One of the most popular techniques is association rule mining (ARM), which is the automatic discovery of pairs of element sets that tend to appear together in a common context. An example would be to discover that the purchase of certain items (say tomatoes and lettuce) in a supermarket transaction usually implies that another set of items (salad dressing) is also bought in that same transaction.

Like other data mining techniques that must process enormous databases, ARM is inherently disk-I/O intensive. These I/O costs can be reduced in two ways: by reducing the number of times the database needs to be scanned, or through parallelization, by partitioning the database between several machines which then perform a distributed ARM (D-ARM) algorithm. In recent years much progress has been made in both directions.

The main task of every ARM algorithm is to discover the sets of items that frequently appear together – the frequent itemsets. The number of database scans required for the task has been reduced from a number equal to the size of the largest itemset in Apriori [3], to typically just a single scan in modern ARM algorithms such as Sampling and DIC [17, 5].

Much progress has also been made in parallelized algorithms. With these, the architecture of the parallel system plays a key role. For instance, many algorithms were proposed which take advantage of the fast interconnect, or the shared memory, of parallel computers. The latest development with these is [18], in which each process makes just two passes over its portion of the database.

Parallel computers are, however, very costly. Hence, although these algorithms were shown to scale up to 128 processors, few organizations can afford to spend such resources on data mining. The alternative is distributed algorithms, which can be run on cheap clusters of standard, off-the-shelf PCs. Algorithms suitable for such systems include the CD and FDM algorithms [2, 6], both parallelized versions of Apriori, which were published shortly after it was described. However, while clusters may easily and cheaply be scaled to hundreds of machines, these algorithms were shown not to scale well [15]. The DDM algorithm [15], which overcomes this scalability problem, was recently described. Unfortunately, all the D-ARM algorithms for share-nothing machines scan the database as many times as Apriori. Since many business databases contain large frequent itemsets (long patterns), these algorithms are not competitive with DIC and Sampling.

In this work we present a parallelized version of the Sampling algorithm, called D-Sampling. The algorithm is intended for clusters of share-nothing machines. The main obstacle of this parallelization, that of achieving a coherent view of the distributed sample at reasonable communication costs, was overcome using ideas taken from DDM. Our distributed algorithm scans the database once, just like the Sampling algorithm, and is thus more efficient than any D-ARM algorithm known today. Not only does this algorithm divide the disk-I/O costs of the single scan by partitioning the database among several machines, but also uses the combined memory to linearly increase the size of the sample. This increase further improves the performance of the algorithm because the safety margin required in Sam-

*This work was supported in part by Microsoft Academic Foundation and by THE ISRAEL SCIENCE FOUNDATION founded by the Israel Academy of Sciences and Humanities.

pling decreases when the (global) sample size increases.

Extensive experiments on standard synthetic benchmarks show that D-Sampling is superior to previous algorithms in every way. When compared to Sampling – one of the best sequential algorithms known today – it offers superlinear speedup. When compared to FDM, it improves runtime by orders of magnitude. Finally, on scalability tests, an increase in both the number of computing nodes and the size of the database does not degrade D-Sampling performance. FDM, on the other hand, suffers performance degradation in these tests.

The rest of this paper is structured as follows: We conclude this section with some notations and a formal definition of the D-ARM problem. In the next section we present relevant previous work. Section 3 describes the D-Sampling algorithm, and section 4 provides the required statistical background. Section 5 describes the experiments we conducted to verify D-Sampling performance. We conclude with some open research problems in section 6.

1.1 Notation and Problem Definition

Let $I = \{i_1, i_2, ..., i_m\}$ be the items in a certain domain. An *itemset* is a subset of I. A *transaction* t is also a subset of I which is associated with a unique transaction identifier – TID. A database DB is a list of such transactions. Let $\overline{DB} = \left\{DB^1, DB^2, ..., DB^n\right\}$ be a partition of DB into n parts. Let S be a list of transactions which were sampled uniformly from DB, and let $\overline{S} = \left\{S^1, S^2, ..., S^n\right\}$ be the partition of S induced by $\overline{DB}$. For any itemset X and any group of transactions A, $Support\left(X, A\right)$ is the number of transactions in A which contain all the items of X and $Freq\left(X, A\right) = \frac{Support(X,A)}{|A|}$. We call $Freq\left(X, DB^i\right)$ the *local frequency* of X in partition i and $Freq\left(X, DB\right)$ its *global frequency*; likewise, we call $Freq\left(X, S^i\right)$ the *estimated local frequency* of X in partition i and $Freq\left(X, S\right)$ its *estimated global frequency*.

For some frequency threshold $0 \leq MinFreq \leq 1$, we say that an itemset X is *frequent* in A if $Freq\left(X, A\right) \geq MinFreq$ and infrequent otherwise. If A is a sample, we say that X is *estimated frequent* or *estimated infrequent*. If A is a partition, we say that X is *locally frequent*, and if A is the whole database, then X is *globally frequent*. Hence an itemset may be estimated locally frequent in the k^{th} partition, globally infrequent, etc. The group of all itemsets with frequency above or equal to fr in A is called $\mathcal{F}_{fr}\left[A\right]$. The *negative border* of $\mathcal{F}_{fr}\left[A\right]$ is all those itemsets which are not themselves in $\mathcal{F}_{fr}\left[A\right]$ but have all their subsets in $\mathcal{F}_{fr}\left[A\right]$. Finally, for a pair of globally frequent itemsets X and Y such that $X \cap Y = \emptyset$, and some confidence threshold $0 < MinConf \leq 1$, we say the rule $X \Rightarrow Y$ is *confident* if and only if $Freq\left(X \cup Y, DB\right) \geq MinConf \cdot Freq\left(X, DB\right)$.

Definition 1 *Given a partitioned database $\overline{DB}$, and given $MinFreq$ and $MinConf$, the D-ARM problem is to find all the confident rules between frequent itemsets in $\overline{DB}$.*

2 Previous Work

Since its introduction in 1993 [1], the ARM problem has been studied intensively. Many algorithms, representing several different approaches, were suggested. Some algorithms, such as Apriori, Partition, DHP, DIC, and FP-growth [3, 14, 11, 5, 8], are bottom-up, starting from itemsets of size 1 and working up. Others, like Pincer-Search [10], use a hybrid approach, trying to guess large itemsets at an early stage. Most algorithms, including those cited above, adhere to the original problem definition, while others search for different kinds of rules [5, 16, 13].

Algorithms for the D-ARM problem usually can be seen as parallelizations of sequential ARM algorithms. The CD, FDM, and DDM [2, 6, 15] algorithms parallelize Apriori [3], and PDM [12] parallelizes DHP [11]. The major difference between parallel algorithms is in the architecture of the parallel machine. This may be shared memory as in the case of [18], distributed shared memory as in [9], or shared nothing as in [2, 6, 15].

One of the best sequential ARM algorithms – Sampling – was presented in 1996 by Toivonen [17]. The idea behind Sampling is simple. A random sample of the database is used to predict all the frequent itemsets, which are then validated in a single database scan. Because this approach is probabilistic, and therefore fallible, not only the frequent itemsets are counted in the scan but also their negative border. If the scan reveals that itemsets that were predicted to belong to the negative border are frequent then a second scan is performed to discover whether any superset of these itemsets is also frequent. To further reduce the chance of failure, Toivonen suggests that mining be performed using some $low\_fr < MinFreq$, and the results reported only if they pass the original $MinFreq$ threshold. He also gives a heuristic which can be used to determine low_fr. The cost of using low_fr is an increase in the number of candidates. The Sampling algorithm and the DIC algorithm (Brin 1997 [5]) are the only single-scan ARM algorithms known today. The performance of the two is thus unrivaled by any other sequential ARM algorithm.

The algorithm presented here combines ideas from several groups of algorithms. It first mines a sample of the database and then validates the result and can, thus, be seen as a parallelization of the Sampling algorithm [17]. The sample is stored in a vertical trie structure that resembles the one in [14, 4], and it is mined using modifications of the DDM [15] algorithm, which is Apriori-based.

3 D-Sampling Algorithm

All distributed ARM algorithms that have been presented until now are Apriori based and thus require multiple database scans. The reason why no distributed form of Sampling was suggested in the six years since its presentation may lie in the communication complexity of the problem. As we have seen, the communication complexity of D-ARM algorithms is highly dependent on the number of candidates and on the noise level in the partitioned database. When Sampling reduces the database through sampling and lowers the $MinFreq$ threshold, it greatly increases both the number of candidates and the noise level. This may render a distributed algorithm useless.

This is the reason that the reduced communication complexity of DDM seems to offer an opportunity. The main idea of D-Sampling is to utilize DDM to mine a distributed sample using low_fr instead of $MinFreq$. After $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$ has been identified, the partitioned database is scanned once in parallel to find the actual frequencies of $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$ and its negative border. Those frequencies can then be collected and rules can be generated from itemsets more frequent than $MinFreq$.

We added three modifications to this scheme. First, although the given DDM is levelwise, here it is executed on a memory resident sample. Thus we could modify DDM to develop new itemsets on-the-fly and calculate their estimated frequency with no disk-I/O. Second, a new method for the reduction of $MinFreq$ to low_fr yielded two additional benefits: it is not heuristic, i.e., our error bound is rigorous, and it produces many less candidates than the rigorous method suggested previously. Third, after scanning the database, it would not be wise to just collect the frequencies of all candidates. Since these candidates were calculated according to the lowered threshold, few of them are expected to have frequencies above the original $MinFreq$. Instead, we run DDM once more to decide which candidates are frequent and which are not. We call the modified algorithm D-Sampling (Algorithm 1).

3.1 Algorithm

D-Sampling begins by loading a sample into memory. The sample is stored in a trie – a lexicographic tree. This trie is the main data structure of D-Sampling and is accessed by all its subroutines. Each node of the trie stores, in addition to structural information (parents, descendants etc.), the list of TIDs of those transactions that include the itemset associated with this node. These lists are initialized from the sample for the first level of the trie; when a new trie node – and itemset – are developed, the TID lists of two of the parent nodes are intersected to create the TID list of the new node.

The first step of D-Sampling is to run a modification of DDM on the distributed sample. Then, in order to set low_fr, the algorithm enters a loop; in each cycle through the loop it calls another DDM derivative called M-Max to mine the next M estimated-frequent itemsets. M is a tunable parameter we set to about 100. After it finds those additional itemsets, D-Sampling reduces low_fr to the estimated frequency of the least frequent one and re-estimates the error probability using a formula described in section 4. When this probability drops below the required error probability, the loop ends. Then D-Sampling creates the final candidate set C by adding to $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$ its negative border.

Algorithm 1 D-Sampling

For node i **out of** n
Input:
$MinFreq$, $MinConf$, DB^i, s, M, δ
Output:
The set of confident associations between globally frequent itemsets
Main:
Set $p\_error \leftarrow 1$,$low\_fr \leftarrow MinFreq$
Load a sample S^i of size s from DB^i into memory
Initialize the trie with all the size-1 itemsets and calculate their TID lists
$\mathcal{F}_{low\_fr}\left[\overline{S}\right] \leftarrow MDDM\left(MinFreq\right)$
While $p\_error > \delta$

1. $\mathcal{F}_{low\_fr}\left[\overline{S}\right] \leftarrow \mathcal{F}_{low\_fr}\left[\overline{S}\right] \cup M\_Max\left(M\right)$
2. Set low_fr to the frequency of the least frequent itemset in $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$
3. Set p_error to the new error bound according to $MinFreq$, low_fr and $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$

Let C be $\mathcal{F}_{low\_fr}\left[\overline{S}\right] \cup Negative\_Border\left(\mathcal{F}_{low\_fr}\left[\overline{S}\right]\right)$
Scan the database and compute $Freq\left(c, DB^i\right)$ for each $c \in C$. Update the frequencies in the trie to the computed ones
Compute $\mathcal{F}_{MinFreq}\left[\overline{DB}\right]$ by running $MDDM\left(MinFreq\right)$, this time with the actual frequencies
If exists $c \in \mathcal{F}_{MinFreq}\left[\overline{DB}\right]$ such that $c \notin \mathcal{F}_{low\_fr}\left[\overline{S}\right]$ (i.e., from negative border) report failure
$Gen\_Rules\left(\mathcal{F}_{MinFreq}\left[\overline{DB}\right], MinConf\right)$

Once the candidate set is established, each partition of the database is scanned exactly once and in parallel, and the actual frequencies of each candidate are calculated. With these frequencies D-Sampling performs yet another round of the modified DDM. In this round the original $MinFreq$ is used; thus, unless there is a failure, this round should never develop a candidate which is outside the negative bor-

der. If indeed no failure occurs, then all frequent itemsets will be evaluated according to the actual frequencies which were found in the database scan. Hence, after this round it is known which of the candidates in C are globally frequent and which are not. In this case, rules are generated from $\mathcal{F}_{MinFreq}\left[\overline{DB}\right]$ using the known global frequencies.

If an itemset belonging to the negative border of $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$ does turn out to be frequent, this means that D-Sampling has failed: a superset of that candidate, which was not counted, might also turn out to be frequent. In this case we suggest the same solution offered by Toivonen: to create a group of additional candidates that includes all combinations of anticipated and unanticipated frequent itemsets, and then perform an additional scan. The size of this group is limited by the number of anticipated frequent itemsets times the number of possible combinations of unanticipated frequent itemsets. Since failures are very rare events, and the probability of multiple failure is exponentially small, the additional scan will incur costs that are of the same scale as the first scan.

3.2 MDDM – A Modified Distributed Decision Miner

The original DDM algorithm, as described in [15], is levelwise. When the database is small enough to fit into memory, the levelwise structure of the algorithm becomes superfluous. Modified Distributed Decision Miner, or MDDM (Algorithm 2), therefore starts by developing all the locally frequent candidates, regardless of their size. It then continues to develop candidates whenever they are required, i.e., when all their subsets are assumed frequent (according to the local hypothesis - P) or when another node refers to the associated itemset.

The remaining steps in MDDM are the same as in DDM. Each party looks for itemsets for which the global hypothesis and local hypothesis disagree and communicate their local counts to the rest of the parties. When no such itemset exists, the party passes (it can return to activity if new information arrives). If all of the parties pass, the algorithm terminates and the itemsets which are predicted to be frequent according to the public hypothesis H are the estimated globally frequent ones. If a message is received for an itemset which has not yet been developed, it is developed on-the-fly and its local frequency is calculated.

3.3 M-Max Algorithm

The modified DDM algorithm identifies all itemsets with frequency above $MinFreq$. D-Sampling, however, requires a further decrease in the frequency of itemsets which are included in the database scan. The reason for this, as we shall see in section 4, is that three parameters affect the

Algorithm 2 Modified Distributed Decision Miner

For node i out of n
Input:
fr – the target frequency
Output:
$\mathcal{F}_{fr}\left[\overline{S}\right]$
Definitions:

$$P\left(X,S^i\right)=\sum_{j\in G(X)}\frac{\left|S^j\right|Freq\left(X,S^j\right)}{|S|}+\sum_{j\notin G(X)}\frac{\left|S^j\right|Freq\left(X,S^i\right)}{|S|}$$

$$H\left(X\right)=\begin{cases}\frac{\sum_{j\in G(X)}\left|S^j\right|Freq\left(X,S^j\right)}{\sum_{j\in G(X)}\left|S^j\right|} & \mathrm{G}\left(X\right)\neq\emptyset\\ 0 & otherwise\end{cases}$$

Main:
Develop all the candidates which are more frequent than fr according to P
Do

- Choose a candidate X that was not yet chosen and for which either $H\left(X\right)<fr\leq P\left(X,S^i\right)$ or $P\left(X,S^i\right)<fr\leq H\left(X\right)$
- Broadcast $m=\left\langle id\left(X\right),Freq\left(X,S^i\right)\right\rangle$
- If no such itemset exists broadcast $\langle pass\rangle$

Until $|Passed|=N$
$R\leftarrow$all X with $H\left(X\right)\geq fr$
Return R
When node i receives a message m from party j:

1. If $m=\langle pass\rangle$ insert j into $Passed$
2. Else $m=\left\langle id\left(X\right),Freq\left(X,S^j\right)\right\rangle$
   If $j\in Passed$ remove j from $Passed$
   If X was not developed then: develop it, set $G\left(X\right)=\emptyset$, Calculate $X.tid\_list$ by intersecting the TID lists of two of X's immediate subsets and set $Freq\left(X,S^i\right)=\frac{|X.tid\_list|}{|S^i|}$
   Insert j to $G\left(X\right)$
   Recalculate $H\left(X\right)$ and $P\left(X,S^i\right)$

chances for failure. These are the size of the sample N, the size of the negative border, and the estimated frequency of the least frequent candidate. The first parameter is given, the second is a rather arbitrary value which we can calculate or bound, and the last parameter is the one we can control.

The frequency of the least frequent candidate can be controlled by reducing low_fr. However, this must be done with care: lowering the frequency threshold increases the number of candidates. This increase depends on the distribution of itemsets in the database and is therefore nondeterministic. The larger number of candidates affects the scan time: the more candidates you have, the more comparisons must be made per transaction. In a distributed setting, the number of candidates is also strongly tied to the communication complexity of the algorithm.

To better control the reduction of low_fr, we propose another version of DDM called M-Max (Algorithm 3). M-Max increases the number of frequent itemsets by a given factor rather than decreasing the threshold value by an arbitrary value. Although worst case analysis shows that an increase of even one frequent itemset may require that any number of additional candidates be considered, the number of such candidates tends to remain small and roughly proportional to the number of additional frequent itemsets. We complement this algorithm with a new bound for the error (presented in section 4). The combined scheme is both rigorous and economical in the number of candidates.

The M-Max algorithm is based on the inference that changing the $MinFreq$ threshold to the H-value of the M-largest itemset[1] every time an itemset is developed or a hypothesis value is changed will result in all parties agreeing on the M most frequent itemsets when DDM terminates. This is easy to prove. Take any final state of the modified algorithm. The H value of each itemset is equal in all parties; hence, the final $MinFreq$ is equal in all parties as well. Now compare this state to the corresponding state under DDM, with the static $MinFreq$ value set to the one finally agreed upon. The state attained by M-Max is also a valid final state for this DDM. Thus, by virtue of DDM correctness, all parties must be in agreement on the same set of frequent itemsets.

As a stand-alone ARM algorithm, M-Max may be impractical because a node may be required to refer to itemsets it has not yet developed. If the database is large, this would require an additional disk scan whenever new candidates are developed. Nevertheless, at the low_fr correction stage of D-Sampling, the database is the memory-resident sample. It is thus possible to evaluate the frequency of arbitrary itemsets with no disk-I/O.

[1] P is used when the M largest H is zero.

Algorithm 3 M-Max

For node i out of n
Input:
low_fr
Output:
The M most frequent itemsets not yet in $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$
Definitions: same as for algorithm 3.2
Let B denote the initial size of $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$, $fr = low\_fr$
Main:
Do

1. call set_fr
2. Choose X that was not yet chosen and for which either $H(X) < fr \leq P\left(X, S^i\right)$ or $P\left(X, S^i\right) < fr \leq H(X)$
   Broadcast $m = \left\langle id(X),\ Freq\left(X, S^i\right)\right\rangle$
3. If no such itemset exists broadcast $\langle pass\rangle$

Until $|Passed| = N$
$R \leftarrow$all X in the trie with $H(X) \geq fr$ which are not in $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$
Return R
When node i receives a message m from party j:

1. If $m = \langle pass\rangle$ insert j into $Passed$
2. Else $m = \left\langle id(X),\ Freq\left(X, S^j\right)\right\rangle$
   If $j \in Passed$ remove j from $Passed$
   If X was not developed then: develop it, set $G(X) = \emptyset$, Calculate $X.tid\_list$ by intersecting the TID lists of two of X's immediate subsets and set $Freq\left(X, S^i\right) = \frac{|X.tid\_list|}{|S^i|}$
   Insert j to $G(X)$
   Recalculate $H(X)$ and $P\left(X, S^i\right)$
   call set_fr

procedure set_fr:
Do M times:

- Select the next most frequent itemset outside $\mathcal{F}_{low\_fr}\left[\overline{S}\right]$ and develop its descendants if they have not been developed yet

Set fr to the H value of the last itemset selected. For itemsets with $H = 0$ consider P instead.

4 Statistical Analysis

The M-Max subroutine requires that we estimate the probability of an error – i.e., an itemset which is actually frequent but appears with frequency of less than the lowered frequency threshold in the sample. An over estimation may result in an unnecessary decrease in low_fr which would result in a larger than required number of candidates. Here we describe the probability bound we have used in our implementation which outperforms the naive Chernoff bound discussed in the original paper which described the Sampling algorithm.

Let $0 < fr < 1$ be the frequency of some arbitrary itemset X in DB. Consider a random sample S of size N from DB. We will assume that transactions in the sample are independent. Hence, the number of rows in S which contain X can be seen as a random variable, $x \sim Bin(N, fr)$.

The frequency of X in N transactions, $s\_fr = x/N$, is an estimate for fr, which improves as N increases. The best-known way to bound the chance that s_fr will deviate from fr is with the Chernoff bound. We use a tighter bound for the case of binomial distributions (see Hagerup and Rub [7]):

$$Pr\left(|fr - s\_fr| > \varepsilon\right) \leq \left[\left(\frac{1-fr}{1-s\_fr}\right)^{1-s\_fr}\left(\frac{fr}{s\_fr}\right)^{s\_fr}\right]^N$$

Lemma 1 *Given a random uniform sample S of N transactions from DB, a frequency threshold $MinFreq$, the lowered frequency threshold low_fr, and the negative border of $\mathcal{F}_{low\_fr}[S]$, denoted NB, the probability $p_{failure}$ that any $X \in NB$ will have frequency larger than or equal to $MinFreq$ (hence causing failure) is bounded by:*

$$|NB| \cdot \left[\left(\frac{1-MinFreq}{1-low\_fr}\right)^{1-low\_fr}\left(\frac{MinFreq}{low\_fr}\right)^{low\_fr}\right]^N$$

For any specific itemset in NB, the probability that this itemset will cause failure is the probability that its estimated frequency is below low_fr while its actual frequency is above $MinFreq$. Substituting $MinFreq$ for fr and low_fr for s_fr, the bound gives us:

$$Pr\left(|Freq\left(X, DB\right) - Freq\left(X, S\right)| > \epsilon\right) \leq$$

$$\left[\left(\frac{1-MinFreq}{1-low\_fr}\right)^{1-low\_fr}\left(\frac{MinFreq}{low\_fr}\right)^{low\_fr}\right]^N$$

As for the entire NB:

$$Pr\left(\exists X \in NB : X\ fails\right) \leq \sum_{X \in NB} Pr\left(X\ fails\right) \leq$$

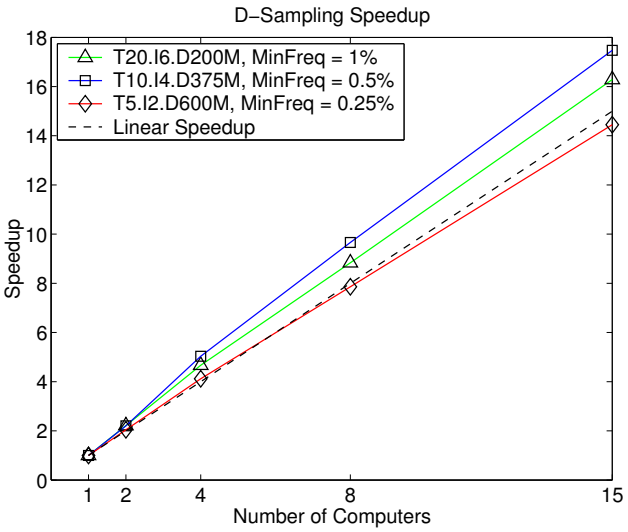

Figure 1. D-Sampling speedup.

$$|NB| \left[\left(\frac{1-MinFreq}{1-low\_fr}\right)^{1-low\_fr}\left(\frac{MinFreq}{low\_fr}\right)^{low\_fr}\right]^N$$

Since calculating the negative border is in itself a costly process, we choose to relax this bound by substituting $|I|\,|\mathcal{F}_{low\_fr}[S]|$ for $|NB|$. Obviously, any itemset in $\mathcal{F}_{low\_fr}[S]$ can only be extended by at most $|I|$ items, and thus this relaxed bound holds.

Corollary 1 ***(Toivonen 1996)*** *If none of the itemsets in the negative border caused failure, then no other itemset can cause failure.*

Any other itemset X outside $\mathcal{F}_{low\_fr}[S]$ and NB must include a subset from NB. Hence its frequency must be less than or equal to the frequency of this subset. It follows that if the frequency of each itemset in NB is below $MinFreq$, so is the frequency of X.

5 Experiments

We carried out three sets of experiments. The first set tested D-Sampling to see how much faster it is to run the algorithm with the database split among n machines than to run it on a single node. The second set compared D-Sampling, DDM and FDM on a range of $MinFreq$ values. The last one checked scale-up: the change in runtime when the number of machines is increased together with the size of the database.

We ran our experiments on two clusters: the first cluster, which was used for the first, second and fourth sets of experiments, consisted of 15 Pentium computers with dual 1.7GHz processors. Each of the computers had at least 1

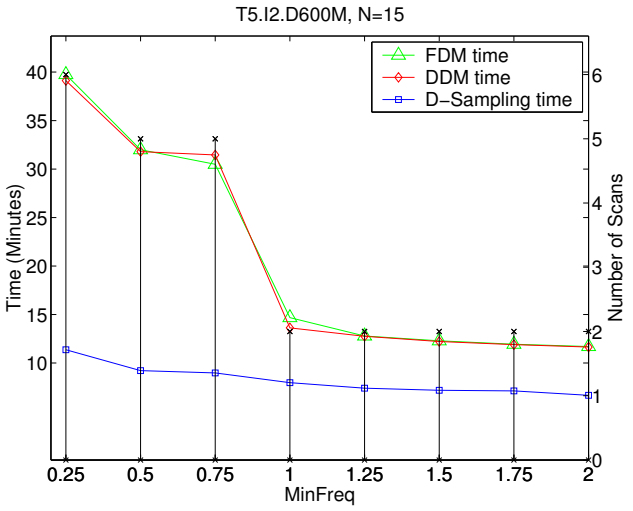

Figure 2. Runtime of D-Sampling, DDM, and FDM for varying MinFreq.

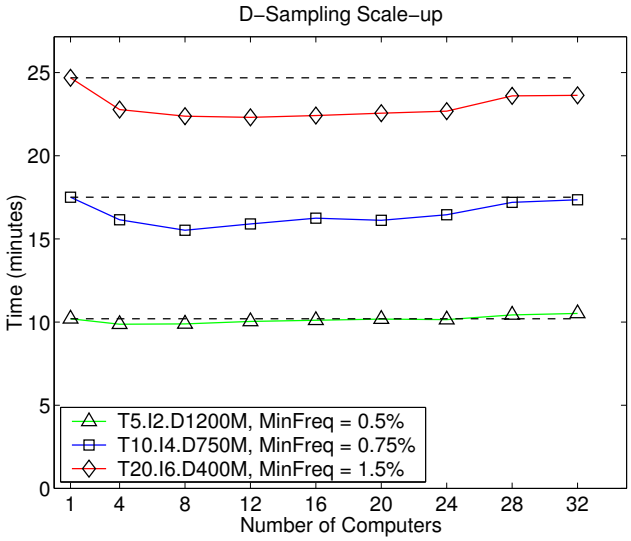

Figure 3. D-Sampling scale-up.

gigabyte of main memory. The computers were connected via an Ethernet-100 network. The second cluster, which we used for the scale-up experiments, was composed of 32 Pentium computers with a dual 500MHz processor. Each computer had 256 megabytes of memory. The second cluster was also connected via an Ethernet-100 network.

All of the experiments were performed with synthetic databases produced by the standard gen tool $available from http : //www.almaden.ibm.com/cs/quest$. The databases were built with the same parameters which were used by Toivonen in [17]. The only change we made was to enlarge the databases to about 18 gigabytes each; had we used the original sizes, the whole database would fit, when partitioned, into the memory of the computers. The database T5.I2.D600M has 600M transactions, each containing an average of five items, and patterns of length two. T10.I4.D375M and T20.I6.D200M follow the same encoding. When the database was to be partitioned, we divided it arbitrarily by writing transaction number TID into the $TID\%n$ partition.

5.1 Speedup Results

The speedup experiments were designed to demonstrate that parallelization works well for Sampling. We thus ran D-Sampling with $n = 1$ (with $n = 1$, D-Sampling reverts to Sampling) on a large database. Then we tested how splitting the database between n computers affects the algorithm's performance.

As figure 1 shows, the basic speedup of D-Sampling is slightly sublinear. However, when the number of candidates is large, the speed-up becomes superlinear. This is because the global sample size increases with the number of computers. This larger sample size translates into a higher low_fr value and thus to a smaller number of candidates than with $n = 1$.

5.2 Dependency on $MinFreq$

The second set of experiments (figure 2) demonstrates the dependency of D-Sampling performance on $MinFreq$, which determines the number and size of the candidates. We compared the D-Sampling runtime to that of both DDM and FDM. D-Sampling turned out to be insensitive to the reduction in $MinFreq$; its runtime increased by no more than 50% across the whole range. On the other hand, the runtime of DDM and FDM increased rapidly as $MinFreq$ is decreased. This is because of the additional scans required as increasingly larger itemsets become frequent. Because it performs just one database scan, D-Sampling is expected to be superior to any levelwise D-ARM algorithm, just as Sampling is superior to all levelwise ARM algorithms.

5.3 Scale-up

The third set of tests was aimed at testing the scalability of D-Sampling. Here the partition size was fixed. We used a database of about 1.5 gigabytes on each computer. A scalable algorithm should have the same runtime regardless of the number of computers.

D-Sampling creates the same communication load per candidate as DDM. However, because it generates more

candidates, it uses more communication. As can be seen from the graphs in figure 3, D-Sampling is scalable in two of the tests. In fact, for mid-range numbers of computers, D-Sampling runs even faster than with $n = 1$; this is due to the superlinear speed-up discussed earlier. The mild slowdown seen in figure 3c is due to the smaller average pattern size and the smaller number of candidates in T5.I2.D1200M. The larger the number of candidates, the greater the saving in candidates when the number of computers increases. If there are enough large patterns, this saving will compensate for the increasing communication overhead. Such is not the case, however, with T5.I2.D1200M.

6 Conclusions and Future Research

We presented a new D-ARM algorithm that uses the communication efficiency of the DDM algorithm to parallelize the single-scan Sampling algorithm. Experiments prove that the new algorithm has superlinear speedup and outperforms both FDM and DDM with any $MinFreq$ value. The exact improvement in relation to previous algorithms depends on the number of database scans they require. Experiments demonstrate good scalability, provided the database scan is the major bottleneck of the algorithm.

Some open questions still remain. First, it would be interesting to continue partitioning the database until every partition becomes memory resident. This approach may lead to a D-ARM algorithm that mines a database by loading it into the memory of large number of computers and then runs with no disk-I/O at all. Second, it would be interesting to have a parallelized version of the other single-scan ARM algorithm – DIC – on a share-nothing cluster**,** or of the two-scans partition algorithm. Finally, we feel that the full potential of the M-Max algorithm has not yet been realized; we intend to research additional applications for this algorithm.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD Int'l. Conference on Management of Data*, pages 207–216, Washington, D.C., June 1993.

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

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l. Conference on Very Large Databases (VLDB'94)*, pages 487 – 499, Santiago, Chile, September 1994.

[4] V. S. Ananthanarayana, D. K. Subramanian, and M. N. Murty. Scalable, distributed and dynamic mining of association rules. In *Proceedings of HiPC'00*, pages 559–566, Bangalore, India, 2000.

[5] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record*, 6(2):255–264, June 1997.

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

[7] T. Hagerup and C. Rub. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305 – 308, 1989/90.

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

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

[10] D.-I. Lin and Z. M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent set. In *Extending Database Technology*, pages 105–119, 1998.

[11] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. of ACM SIGMOD Int'l. Conference on Management of Data*, pages 175 – 186, San Jose, California, May 1995.

[12] J. S. Park, M.-S. Chen, and P. S. Yu. Efficient parallel data mining for association rules. In *Proc. of ACM Int'l. Conference on Information and Knowledge Management*, pages 31 – 36, Baltimore, MD, November 1995.

[13] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *Proc. of the ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 350–354, Boston, MA, 2000.

[14] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. *The VLDB Journal*, pages 432–444, 1995.

[15] A. Schuster and R. Wolff. Communication-efficient distributed mining of association rules. In *Proc. of the 2001 ACM SIGMOD Int'l. Conference on Management of Data*, pages 473 – 484, Santa Barbara, California, May 2001.

[16] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of the 20th Int'l. Conference on Very Large Databases (VLDB'94)*, pages 407 – 419, Santiago, Chile, September 1994.

[17] H. Toivonen. Sampling large databases for association rules. In *The VLDB Journal*, pages 134–145, 1996.

[18] O. R. Zaiane, M. El-Hajj, and P. Lu. Fast parallel association rules mining without candidacy generation. In *IEEE 2001 International Conference on Data Mining (ICDM'2001)*, pages 665–668, 2001.