

Efficient, Accurate and Privacy-Preserving Data Mining for Frequent Itemsets in Distributed Databases

Adriano A. Veloso^{1,2} Wagner Meira Jr.¹
Srinivasan Parthasarathy² Márcio Bunte de Carvalho¹

¹Computer Science Department – Universidade Federal de Minas Gerais
Belo Horizonte – MG – Brazil {adrianov,meira,mlbc}@dcc.ufmg.br

²Computer and Information Science Department The Ohio-State University –
Columbus – OH – USA srini@cis.ohio-state.edu

Abstract

Mining distributed databases is emerging as a fundamental computational problem. A common approach for mining distributed databases is to move all of the data from each database to a central site and a single model is built. This approach is accurate, but too expensive in terms of time required. For this reason, several approaches were developed to efficiently mine distributed databases, but they still ignore a key issue – privacy. Privacy is the right of individuals or organizations to keep their own information secret. Privacy concerns can prevent data movement – data may be distributed among several custodians, none of which is allowed to transfer its data to another site.

In this paper we present an efficient approach for mining frequent itemsets in distributed databases. Our approach is accurate and uses a privacy-preserving communication mechanism. The proposed approach is also efficient in terms of message passing overhead, requiring only one round of communication during the mining operation. We show that our privacy-preserving distributed approach has superior performance when compared to the application of a well-known mining algorithm in distributed databases.

1 Introduction

Although data mining has its roots in the traditional fields of machine learning and statistics, it is the sheer volume of data that poses the most serious problem to its application. Many organizations already have warehouses that store data amounts in the terabyte range, and mining these warehouses requires both large memory and disk space, and high speed computing.

Traditional methods [2, 6, 11, 14] typically made the assumption that the data is centralized and memory resident. This assumption is no longer tenable. A direct application of traditional mining algorithms to distributed databases is not effective, because it requires a large amount of communication overhead. Implementation of data mining ideas in high-performance distributed computing environments is thus becoming crucial for ensuring system scalability. However, distributed mining has implications far beyond just scaling up algorithms. This is because sometimes the problem is not simply that the data is distributed, but that it must remain distributed. The issues concerning modern organizations are not only the size of the data to be mined, but also its distributed nature. Modern organizations may have their databases logically and physically located at different (distant) places, or they may be willing to share their data mining models, but not their data. In all these cases what is needed is a decentralized approach to data mining.

In this paper we are interested in the problem of mining frequent itemsets in distributed databases. Mining frequent itemsets is a fundamental and essential task in many data mining applications. These applications include the discovery of association rules, strong rules, correlations, sequential rules, episodes, multi-dimensional patterns, and several other important discovery tasks. We consider three important issues when mining frequent itemsets in distributed databases: performance, accuracy, and limited access to possibly privacy sensitive data. Developing efficient techniques to mine distributed data is a challenging task since these issues are typically contradictory. This is because data mining techniques require accurate input data for their results to be meaningful, but performance and privacy concerns may influence the input data (i.e., sampling [10, 13], data perturbation [3, 5], cryptography [9], etc.).

Let us consider a real-life motivating example where performance, accuracy and privacy are important issues. Credit card transactions is taking a large share of the payment systems worldwide, and have led to a higher rate of stolen account numbers and subsequent losses by banks. Each bank has a database which stores both legitimate and fraudulent credit card transactions, so let us consider distributed data mining for credit card fraud detection. The objective is to build a data mining model which will be used by automated fraud detector systems to prevent fraudulent transactions. This data mining model must be very general and accurate, because a

mistake means great loss of money. In order to build accurate models of credit card fraud, a data mining system must have access to information about the fraudulent patterns of all banks (i.e., some types of fraud can have occurred only in one bank). The problem is that banks are restricted by law (and also by competitive reasons) from sharing data about their customers with other banks. However, they may share “black-box” models; that is, they can share knowledge about fraudulent transactions, but not their data. Furthermore, there are millions of credit card transactions processed each day, and the data mining system must be scalable and able to compute the model of credit card fraud in a timely manner.

We present an efficient approach for mining frequent itemsets in *horizontally-distributed* databases. Instead of sharing possibly privacy sensitive data to perform the distributed mining task, we chose to share just a small portion of each local model, which are used to construct the global model of frequent itemsets. This choice also makes our approach extremely efficient in terms of communication overhead, enabling it to be used for mining even geographically distributed databases. We also developed a communication mechanism to ensure privacy among the sites involved in the mining operation. We prove that the global model generated by our approach is totally accurate, we present bounds for the total amount of communication, and we demonstrate the performance of our approach through a set of experiments under a variety of conditions.

1.1 Frequent Itemset Mining

The frequent itemset mining task can be stated as follows: Let \mathcal{I} be a set of distinct attributes, also called items. Let \mathcal{D} be a database of transactions, where each transaction has a unique identifier (*tid*) and contains a set of items. A set of items is called an *itemset* where for each nonnegative integer k , an itemset with exactly k items is called a k -itemset. The *tidset* of an itemset C corresponds to the set of all transaction identifiers (*tids*) where the itemset C occurs. The *support count* of C , is the number of transactions of \mathcal{D} in which it occurs as a subset. Similarly, the *support* of C is the percentage of transactions of \mathcal{D} in which it occurs as a subset. The itemsets that meet a user specified *minimum support* are referred to as *frequent* itemsets. A frequent itemset is *maximal* if it is not subset of any other frequent itemset.

The database \mathcal{D} can be divided into n partitions, d_1, d_2, \dots, d_n . Each partition d_i is assigned to a site S_i . We say that \mathcal{D} is horizontally distributed if its transactions are distributed among the sites. In this case, let $C.sup$ and $C.sup_i$ be the respective support counts of C in \mathcal{D} and d_i . We will call $C.sup$ the *global support count* of C , and $C.sup_i$ the *local support count* of C in d_i . For a given minimum support s , C is *global frequent* if $C.sup \geq s \times |\mathcal{D}|$; correspondingly, C is *local frequent* at d_i , if $C.sup_i \geq s \times |d_i|$. The set of all maximal global frequent itemsets is

denoted as $MFI_{\mathcal{D}}$, and the set of maximal local frequent itemsets at d_i is denoted as MFI_{d_i} . The task of mining frequent itemsets in distributed databases is to find all global frequent itemsets.

1.2 Related Work

A common approach for mining distributed databases is the centralized one, where all the data is moved to a single central location and then mined. Another common approach is the local one, where models are built locally in each site, and then moved to a common location where they are combined. The later approach is the quickest but often the least accurate, while the former approach is more accurate but generally quite expensive in terms of time required. In the search for accurate and efficient solutions, several intermediate approaches have been proposed [1, 4, 15]. In [1] three distributed mining approaches were proposed. The COUNT DISTRIBUTION algorithm is a simple parallel implementation of APRIORI [2]. All sites generate the entire set of candidates, and each site can thus independently get local support counts from its partition. At each iteration the algorithm does a sum reduction operation to obtain the global support counts by exchanging local support counts with all other sites. Since only the support counts are exchanged among the sites, the communication overhead is reduced. However, it performs one round of communication per iteration (note that synchronization is implicit in communication). The DATA DISTRIBUTION algorithm generates disjoint candidate sets on each site. However, to generate the global support counts, each site has to scan the entire database (its local partition and all remote ones) in all iterations of the algorithm. Hence this approach suffers from high communication overhead. The CANDIDATE DISTRIBUTION algorithm partitions the candidates during each iteration, so that each site can generate disjoint candidates independently of the other sites, but it still requires one round of communication per iteration. In [15] two distributed algorithms were presented, PARECLAT and PARMAXECLAT. Both algorithms are based on the concept of equivalence classes. Each equivalence class corresponds to a sub-tree in the search space for frequent itemsets, and they can be processed asynchronously on each site. PARECLAT outperforms DATA, COUNT, and CANDIDATE DISTRIBUTION algorithms for more than one order of magnitude. PARMAXECLAT outperforms PARECLAT because it searches only the maximal frequent itemsets, instead of all frequent itemsets.

These techniques are devised to scale up a given algorithm (i.e., APRIORI, ECLAT, etc.). The data is distributed (or in some cases, replicated) among different sites and a data mining algorithm is executed in parallel on each site. These approaches do not take into account the possible distributed nature of the data. Some assume a high-speed network environment and perform excessive communication operations. These approaches are not efficient when the databases are really geographically distributed. They also assume that each site can have total

access to data from other sites, but this is not possible when mining privacy-sensitive distributed data. Performance and scalability are still important issues, but ignoring the distributed nature of data can make distributed data mining vulnerable to misuse.

Our distributed mining approach does not assume a high-speed network environment, because it is efficient in terms of communication overhead, requiring only one round of message passing and one reduction operation to aggregate final results. This low degree of communication enables our approach to possibly mine even physically (distant) distributed databases. Furthermore, previous work in privacy-preserving data mining [3, 5, 9] has basically addressed one issue: preserving privacy by distorting the data values. The idea is that the distorted data does not reveal private information, and thus it is safe to use for mining. The data distortion approach addresses a different problem from our work. The assumption with distortion is that the values must be kept private from whoever is doing the mining operation. We instead assume that each site is allowed to see its local data, but not the data from the other sites. In return, we are able to get exact, rather than approximate results. In [7] the authors presented a privacy-preserving distributed data mining approach that is accurate, but it requires much more communication than our approach (i.e., one round per iteration).

In the next section we describe the basic principles and definitions necessary to better understand our approach, which is presented in Section 3. In Section 4 we present our experimental results. Finally, in Section 5 we conclude our work.

2 Efficient Distributed Data Mining for Frequent Itemsets

In this section we present our approach for mining distributed databases. We start by describing the basic algorithm in its serial version. Next we show how this algorithm can be efficiently extended to mine distributed databases, and then we prove that the extended algorithm generates an accurate global model of frequent itemsets. Finally, we show a simple privacy-preserving communication mechanism, and present an upper bound for the amount of communication necessary in the mining operation.

2.1 Efficiently Mining Frequent Itemsets in Centralized Databases

Almost all algorithms for mining frequent itemsets use the same procedure – first a set of candidates is generated, next infrequent ones are pruned, and only the frequent ones are used to generate the next set of candidates. Clearly, an important issue in this task is to reduce the number of candidates generated. An interesting approach to reduce the number of candidates

is to first mine the $MFI_{\mathcal{D}}$. Once the $MFI_{\mathcal{D}}$ is found, it is straightforward to obtain all frequent itemsets (and their support counts) in a single database scan, without generating infrequent (and unnecessary) candidates. This approach works because the downward closure property (all subsets of a frequent itemset must be frequent). The number of candidates generated to find the $MFI_{\mathcal{D}}$ is much smaller than the number of candidates generated to directly find all frequent itemsets. Maximal frequent itemsets were successfully used in several mining tasks, including mining dense databases [6], and incremental mining of evolving databases [11, 12, 13].

An efficient search for maximal frequent itemsets must have the ability to quickly remove large branches of the search space from consideration. This property is associated with the number of candidates generated in the search. The smaller the number of candidates generated, the faster the search will be. In [6] an efficient algorithm called GENMAX was proposed. It employs a backtracking search to find the maximal frequent itemsets. Backtracking algorithms are useful for many combinatorial problems where the solution can be represented as a set $I = \{i_0, i_1, \dots\}$, where each i_j is chosen from a finite *possible set*, P_j . Initially I is empty; it is extended one item at a time, as the search space is traversed. The length of I is the same as the depth of the corresponding node in the search tree. Given a k -candidate itemset, $I_k = \{i_0, i_1, \dots, i_{k-1}\}$, the possible values for the next item i_k comes from a subset $R_k \subseteq P_k$ called the *combine set*. If $y \in P_k - R_k$, then nodes in the subtree with root node $I_k = \{i_0, i_1, \dots, i_{k-1}, y\}$ will not be considered by the backtracking algorithm.

Each iteration of the algorithm tries extending I_k with every item x in the combine set R_k . An extension is valid if the resulting itemset I_{k+1} is frequent and is not a subset of any already known maximal frequent itemset. The next step is to extract the new possible set of extensions, P_{k+1} , which consists only of items in R_k that follow x . The new combine set, R_{k+1} , consists of those items in the possible set that produce a frequent itemset when used to extend I_{k+1} . Any item not in the combine set refers to a pruned subtree. The backtrack search performs a depth-first traversal of the search space.

The support computation employed by GENMAX is based on the associativity of itemsets, which is defined as follows. Let C be a k -itemset of items $C_1 \dots C_k$, where $C_i \in I$. Let $\mathcal{L}(C)$ be its tidset and $|\mathcal{L}(C)|$ is the length of $\mathcal{L}(C)$ and thus the support count of C . According to [6], any itemset can be obtained by joining its atoms (individual items) and its support count can be obtained by intersecting the tidsets of two of its subsets.

2.2 Efficiently Mining Frequent Itemsets in Distributed Databases

The search for the $MFI_{\mathcal{D}}$ employed by GENMAX is very efficient, but it can only be applied when \mathcal{D} is a centralized database. Now we will explain how we can extend GENMAX to de-

velop an efficient distributed mining algorithm. We first present Lemma 1, which is the basic theoretical foundation of our approach.

Lemma 1 – A global frequent itemset must be local frequent in, at least, one partition [8].

■

In the first step each site S_i independently performs a search for MFI_{d_i} on its database d_i . After all sites finish their searches, the result will be the set of all local MFIs, $\{\text{MFI}_{d_1}, \text{MFI}_{d_2}, \dots, \text{MFI}_{d_i}\}$. This information is sufficient for determining all local frequent itemsets, and from Lemma 1, it is also sufficient for determining all global frequent itemsets.

When all local MFIs are found, we start the second step. Each site sends its local MFI to a combiner, and then the combiner joins all local MFIs. The result is the set $\bigcup_{i=1}^n \text{MFI}_{d_i}$, which is an upper bound for $\text{MFI}_{\mathcal{D}}$. The combiner then sends this upper bound to all sites.

In the third step each site independently performs a top down enumeration of the potentially global frequent itemsets, as follows. Each itemset present in the upper bound $\bigcup_{i=1}^n \text{MFI}_{d_i}$ is broken into k subsets of size $(k - 1)$. The support count of this itemset is computed by intersecting the tidsets of its atoms. This process iterates generating smaller subsets and computing their support counts until there are no more subsets to be checked. At the end of this step, each site will have the same set of potentially global frequent itemsets (and the support associated with each of these itemsets).

Lemma 2 – $\bigcup_{i=1}^n \text{MFI}_{d_i}$ determines all global frequent itemsets. *Proof.* – We know from Lemma 1 that if C is a global frequent itemset, so it must be local frequent in at least one partition, d_i . If C is local frequent in some partition d_l , so it must be determined by MFI_{d_l} , and consequently by $\bigcup_{i=1}^n \text{MFI}_{d_i}$. □

By Lemma 2 all global frequent itemsets were found, but not all itemsets generated in the third step are global frequent (some of them are just local frequent). The last step makes a reduction operation on the local support counts of each itemset, to verify which of them are global frequent itemsets. The process starts with site S_1 , which sends the support counts of its itemsets (generated in the third step) to site S_2 . Site S_2 sums the support count of each itemset (generated in the third step) with the value of the same itemset obtained from site S_1 , and sends the result to site S_3 . This procedure continues until site S_n has the global support counts of all potentially global frequent itemsets. Then site S_n finds all itemsets that have support greater than or equal to the minimum support threshold, and all global frequent itemsets were found.

We illustrate all steps of the algorithm execution in Figure 1. The value of minimum support is 50%. Site S_1 and site S_2 will perform the distributed mining operation on databases d_1 and

d_2 . In the first step each site mines its local MFI. The result is $MFI_{d_1} = \{ABDE, BCE\}$, and $MFI_{d_2} = \{ACDE, BCD\}$. In the next step, each site sends their local MFIs to the combiner, so that it can compute the upper bound $\bigcup_{i=1}^n MFI_{d_i}$, which is $\{ABDE, BCE, ACDE, BCD\}$. Now, each site computes the support count of each subset of each itemset in $\bigcup_{i=1}^n MFI_{d_i}$. Some of the generated subsets at site S_i are not local frequent in d_i , but their support count must be computed because some of them must be local frequent in other site, and therefore they can still be global frequent itemsets (i.e., ABE is not locally frequent in d_2 , but it is locally frequent in d_1 , and is globally frequent). In the last step the global frequent itemsets are found by aggregating (sum reduction operation) the local counts of each local frequent itemset.

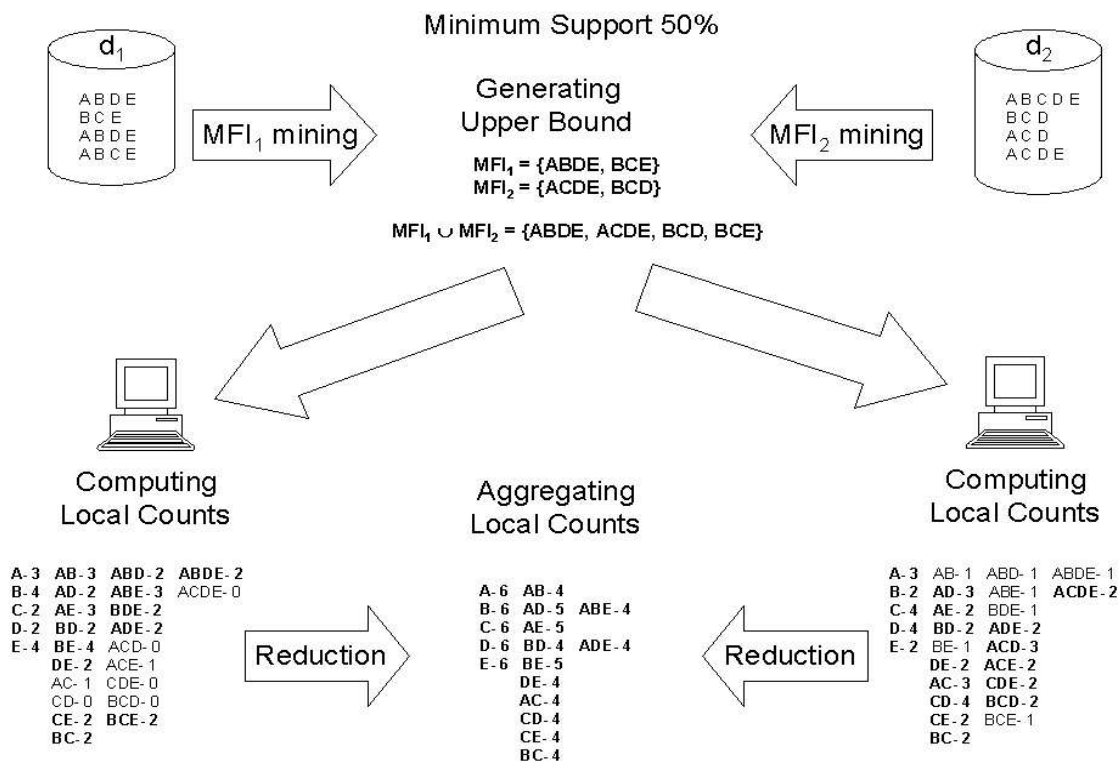


Figure 1: Overall Process of Distributed Mining

2.3 Privacy-Preserving Communication Mechanism

As demonstrated, our distributed algorithm generates a totally accurate model of global frequent itemsets. However, it is not a privacy-preserving approach, because, during the second step, the combiner gets to know the MFI information of other sites, and during the last step site S_2 gets to know the exact local supports of each itemset from site S_1 . To solve this problem we propose

a privacy-preserving communication mechanism. This mechanism simply changes steps 2 and 4 in the following way.

In the new step 2, each site uses another representation of the itemsets (ABD becomes, for instance, 0-14-36) in a way such that the combiner is not able to identify the itemsets. Each site sends its coded local MFI information to the combiner (we assume that all sites use the same codification), which is able to handle this information without knowing the true identity of each item. After combining the local MFIs, the combiner sends the upper bound to all sites, and each site is able to restore the original itemsets codification. At this point of the computation, each site only knows its local MFI information and the upper bound, but not the MFIs from the other sites. In this step the combiner plays the role of a trusted third party.

In step 4, before sending its local support counts to site S_2 , site S_1 generates a random number for each of its itemsets. This random number is then added to the support count of each itemset. With this perturbation, site S_1 can safely send its local support counts, since this information is meaningless to site S_2 . The algorithm continues the same way as before up to the last iteration. After receiving the values of the total local counts, site S_n requests from site S_1 , the values of random numbers and their respective itemsets. Site S_n simply decrement each global support count by the respective number, and checks which itemsets are global frequent. Clearly, this communication mechanism is capable of finding the global frequent itemsets without revealing which itemset belongs to which site.

2.3.1 Discussions

Note that our mechanism is not suitable for some particular situations, that occur when the private information can be revealed by knowledge of one's own data and the final result:

1. There are only two sites: In this case, having the global model and one local model turns possible to derive the other local model.
2. The global support of an itemset is 100%: In this case all sites will know that this itemset and all its subsets also occur in all transactions of the other sites.

2.4 An Upper Bound for the Amount of Communication

We also present an upper bound for the amount of communication performed during the distributed mining operation. Our calculation is based just on the local MFIs and on the size of the upper bound for $\text{MFI}_{\mathcal{D}}$. We divide the upper bound calculation into two steps. The first step is related to the local MFI exchange operation. Since each one of the n sites will have to send its

local MFI to the combiner, the first term is given by: $\sum_{i=1}^n \sum_{j=1}^{|MFI_{d_i}|} |C_{i,j}|$, where $|C_{i,j}|$ is the size of the j^{th} itemset of the local MFI of site S_i . Also, the combiner has to return to all n sites the upper bound for the global MFI, and the amount of communication this operation will need is given by: $n \times \sum_{i=1}^{|UB|} |C_i|$, where UB is the upper bound for $MFI_{\mathcal{D}}$ (i.e., $\bigcup_{i=1}^n MFI_{d_i}$), and $|C_i|$ is the size of the i^{th} itemset in UB.

The second step is related to the local support count reduction operation. In this operation $n - 1$ sites have to pass their local support counts. The amount of communication for this operation is given by: $(n - 1) \times \sum_{i=1}^{|UB|} (2^{|C_j|} - 1)$, where the term $\sum_{i=1}^{|UB|} (2^{|C_j|} - 1)$ represents the local support counts of all subsets of all itemsets in UB.

In our data structure a k -itemset is represented by a set of k integers (of 4 bytes). So, in the worst case (where each itemset is subset of only one maximal frequent itemset), the total amount of communication is given by: $(\sum_{i=1}^n \sum_{j=1}^{|MFI_{d_i}|} |C_{i,j}| + n \times \sum_{i=1}^{|UB|} |C_i| + (n - 1) \times \sum_{i=1}^{|UB|} (2^{|C_j|} - 1)) \times 4$ bytes. This upper bound shows that our approach is extremely efficient in terms of communication overhead, when compared with the amount of communication necessary to transfer all data among the sites.

Now that we demonstrated that our approach is accurate and privacy-preserving, and present an upper bound for the amount of communication necessary in the mining operation, we will evaluate our approach in terms of performance.

3 Experimental Evaluation

Before we present the experimental results, we describe our databases and the computational setup used for distributed mining. Then we present the results on these databases.

3.1 The Databases

We chose several real and synthetic databases for testing the performance of our algorithm. The WCup database is generated from the click-stream log of the 1998 World Cup Web site, which is publicly available at <ftp://researchsmp2.cc.vt.edu/pub/worldcup/>. We scanned the log and produced a transaction file, where each transaction is a session of access to the site by a client. Each item in the transaction is a web request. Not all web requests were turned into items; to become an item, the request must have three properties: (1) Request method is GET; (2) Request status is OK; and (3) File type is HTML. A session starts with a request that satisfies the above properties, and ends when the last click from the client timeouts. The timeout was set as 30 minutes. All requests in a session must come from the same client. We also chose a few

synthetic databases (also available from IBM Almaden), which have been used as benchmarks for testing previous mining algorithms [2]. Table 1 shows the characteristics of the real and synthetic databases used in our evaluation.

Database	#Items	Avg. Length	#Transactions	Size
T10I6D4000K	2,000	10	4,000,000	897MB
T10I8D4000K	2,000	10	4,000,000	862MB
WCup	5,271	8	6,525,879	545MB

Table 1: Database Characteristics.

3.2 The Experimental Environment

All the experiments were performed on a cluster of 8 dual 1GHz processors with 1 GB main memory each. All the nodes are inter-connected via the Myrinet. Each node also has a 60GB local disk attached to it. All the partitioned databases reside on the local disks of each node.

3.3 The Results

3.3.1 Performance Comparison

In order to show the advantages of distributed mining in terms of parallelism, we compare the performance of our algorithm against a sequential algorithm, ECLAT [14]. Both algorithms generate the same set of frequent itemsets (i.e., all global frequent itemsets). In these experiments we varied the number of nodes (i.e., sites) and the minimum support threshold. The basic metric employed was the total execution time to perform the distributed mining operation. Figure 2 shows the execution times of the algorithms on different databases. As we can see even when there is only one node, our approach (here called DMFI) is superior. The reason is that ECLAT generates much more candidates than DMFI. This result shows that applying a search for the maximal frequent itemsets and then generating only frequent subsets is a very efficient approach to reduce the number of candidates generated. In terms of parallel gains, if we look at the best result of DMFI, we can see an improvement of a factor of 9 over ECLAT.

3.3.2 Parallelism: Speedup and Scaleup

We also investigated the performance of our algorithm in speedup and scaleup experiments. In the speedup experiment, we investigated the speedup on a fixed size database with increasing number of nodes. Each database has 8 partitions, and we combine them to form databases with

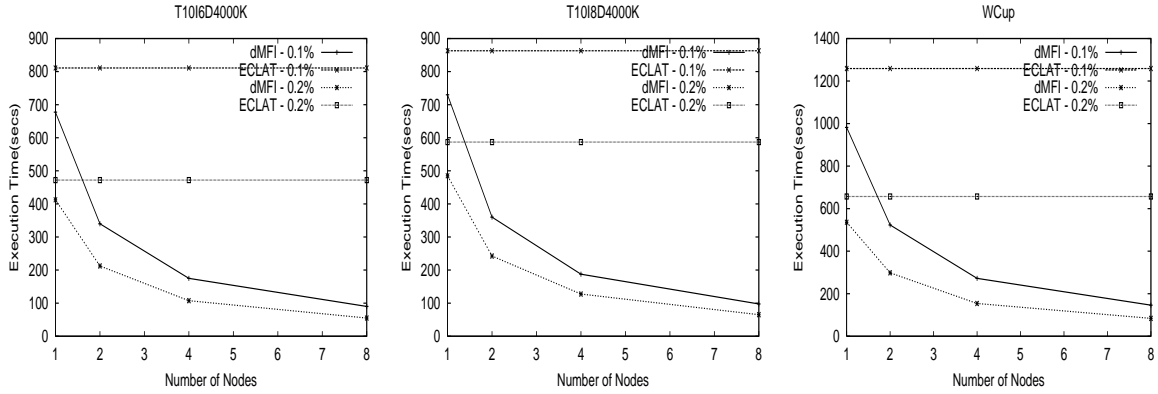


Figure 2: Total Execution Time Comparison.

1, 2, 4, and 8 partitions. With this configuration we performed experiments on 1, 2, 4, and 8 nodes. Figure 3 shows the speedup on the different databases and parallel configurations. The speedup numbers for the real database are not so impressive as the speedup numbers for the synthetic ones. The reason is that the real database has highly skewed data, and therefore each partition of the real database has a very different set of frequent itemsets (and therefore very different local MFIs). On the other hand, the skewness of the synthetic data is very low (i.e., the frequent itemsets are evenly distributed among the partitions), therefore each partition of the synthetic databases is likely to have a similar set of frequent itemsets. From the experiments with synthetic databases we observed that $\bigcup_{i=1}^n \text{MFI}_{d_i}$ (i.e., the upper bound) is very close to $\text{MFI}_{\mathcal{D}}$. This means that the set of local frequent itemsets is very similar to the set of global frequent itemsets, and therefore few infrequent candidates are generated by each node. In order to verify how the quality of the upper bound varies with the number of nodes, we compared the size of the upper bound with the size of $\text{MFI}_{\mathcal{D}}$. Figure 4 shows the relative size of the upper bound for different number of nodes. When there is only one node, the relative size is 1 (i.e., the upper bound is exactly $\text{MFI}_{\mathcal{D}}$), and as expected, it grows with the number of nodes. The growth is greater in the real database and for larger minimum support values. Although better results were achieved with synthetic data (as is always expected), our approach was also efficient for mining real data, achieving significant speedups of almost 7 when using 8 nodes (i.e., 88% of parallel efficiency).

In the scaleup experiments, we employed databases with different number of transactions. The number of transactions ranges from 500,000 to 4,000,000. The number of nodes involved were also increased from 1 to 8 correspondingly. Both the number of transactions and number of nodes are scaled up proportionally. From Figure 5 we can see that our approach is able to keep its execution time almost constant when both the number of transactions and number of nodes

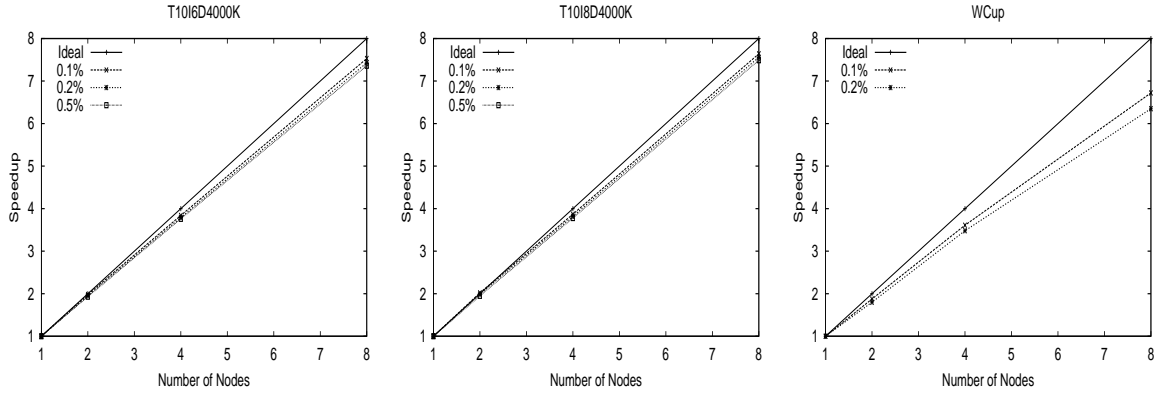


Figure 3: Speedup Performance.

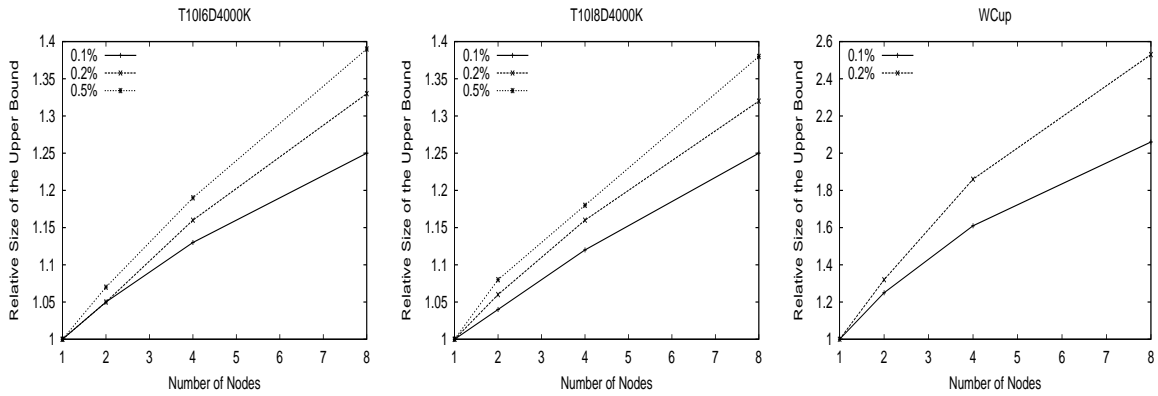


Figure 4: Relative Size of the Upper Bound.

increase linearly, showing that our approach has small amount of communication overhead.

4 Conclusions and Future Work

This paper introduced a novel algorithm for mining frequent itemsets in distributed databases. Instead of moving data or large portions of the model among the sites, we chose to move only a very small portion of each local model, the local MFIs. This choice turns our algorithm extremely efficient in terms of communication overhead, requiring only one round of message passing and one reduction operation to aggregate final results. We prove that the model of global frequent itemsets generated by our algorithm is totally accurate, showed an efficient mechanism to ensure privacy on the communication between the sites, and present an upper bound for the amount of necessary communication.

We compared our approach against a well-known sequential algorithm (but we intend to

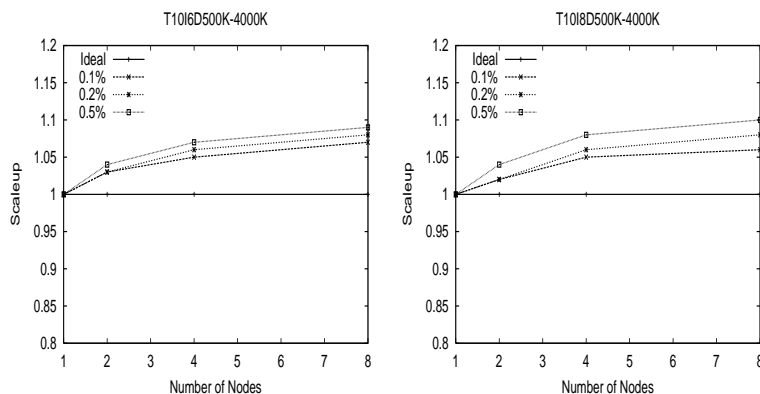


Figure 5: Scaleup Performance.

compare our approach against other distributed algorithms). Experimental results indicate that significant performance improvements can be obtained using our approach. The efficiency in terms of communication can enable our approach to mine really geographically (distributed) databases. Besides performance and scalability, the distributed data mining task exhibits technical problems, like skewed distributions. We are currently working on an alternative approach that is less sensitive to data skewness. Preliminary applications of our approach to mine real databases from actual applications show promising results.

References

- [1] R. Agrawal and J. Shafer. Parallel mining of association rules. In *IEEE Transactions on Knowledge and Data Engineering*, volume 8, pages 962–969, December 1996.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conf. on Very Large Databases*, SanTiago, Chile, June 1994.
- [3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450, May 2000.
- [4] D. Cheung, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. of the 4th Int'l Conference on Parallel and Distributed Systems*, pages 31–42, Los Alamitos, USA, December 1996.
- [5] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, July 2002.

- [6] K. Gouda and M. Zaki. Efficiently mining maximal frequent itemsets. In *Proc. of the 1st IEEE Int'l Conference on Data Mining*, San Jose, USA, November 2001.
- [7] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, June 2002.
- [8] J. Lin and M. Dunham. Mining association rules: Anti-skew algorithms. In *Proc. of 14th IEEE Int'l Conf. on Data Engineering*, October 1998.
- [9] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Advances in Cryptology*, 1880:36–54, 2000.
- [10] A. Veloso, W. Meira Jr., and M. B. de Carvalho. Mining reliable models of associations in dynamic databases. In *Proc. of the 17th Brazilian Symposium on Databases*, pages 263–277, Gramado, Brazil, October 2002.
- [11] A. Veloso, W. Meira Jr., M. B. de Carvalho, S. Parthasarathy, and M. Zaki. Parallel, incremental and interactive mining for frequent itemsets in evolving databases. In *Proc. of the 6th Int'l Workshop on High Performance, Pervasive and Stream Data Mining*, San Francisco, USA, April 2003.
- [12] A. Veloso, W. Meira Jr., M. B. de Carvalho, B. Pôssas, S. Parthasarathy, and M. Zaki. Mining frequent itemsets in evolving databases. In *Proc. of the 2nd SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.
- [13] A. Veloso, W. Meira Jr., M. B. de Carvalho, B. Rocha, S. Parthasarathy, and M. Zaki. Efficiently mining approximate models of associations in evolving databases. In *Proc. of the 6th Int'l Conf. on Principles and Practices of Data Mining and Knowledge Discovery in Databases*, Helsinki, Finland, August 2002.
- [14] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of the 3rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, August 1997.
- [15] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New parallel algorithms for fast discovery of association rules. *Data Mining and Knowledge Discovery: An International Journal*, 4(1):343–373, December 1997.