Mining Frequent Itemsets in Distributed and Dynamic Databases

Matthew Eric Otey Srinivasan Parthasarathy Chao Wang
Computer and Information Science Dept.
The Ohio State University
{otey, srini, wachao}@cis.ohio-state.edu

Adriano Veloso Wagner Meira Jr.
Computer Science Dept.
Universidade Federal de Minas Gerais
{adrianov, meira}@dcc.ufmg.br

Abstract

Traditional methods for data mining typically make the assumption that the data is centralized, memory-resident and static. This assumption is no longer tenable. Such methods waste computational and I/O resources when data is dynamic, and they impose excessive communication overhead when data is distributed. Efficient implementation of incremental data mining methods is thus becoming crucial for ensuring system scalability and facilitating knowledge discovery when data is dynamic and distributed. In this paper we address this issue in the context of the important data mining task of frequent itemset mining. We first present an efficient algorithm which dynamically maintains the required information even in the presence of data updates without examining the entire dataset. We then show how to parallelize this incremental algorithm. We also propose a distributed asynchronous algorithm, which imposes minimal communication overhead for mining distributed dynamic datasets. Our distributed approach is capable of generating local models (in which each site has a summary of its own database) as well as the global model of frequent itemsets (in which all sites have a summary of the entire database). This ability permits our approach not only to generate frequent itemsets, but also high-contrast frequent itemsets, which allows one to examine how the data is skewed over different sites.

Index Terms

Incremental data mining, parallel computing, distributed computing, grid computing.

I. Introduction

The field of knowledge discovery and data mining (KDD), spurred by advances in data collection technology, is concerned with the process of deriving interesting and useful patterns from large datasets. Frequent itemset mining is a core data mining task. It has an elegantly

This work is the result of a collaboration between Ohio State University and Universidade Federal de Minas Gerais. This work was done while Adriano Veloso was visiting Ohio State University. This work was funded in part by the NSF.

simple problem statement: to find the set of all subsets of items that frequently occur together in database records or transactions. Although this task has a simple statement, it is CPU and I/O intensive, mainly because the large number of itemsets that are typically generated and the large size of the datasets involved in the process.

Consider the problem of mining frequent itemsets from a dynamic dataset, such as those found in the domains of e-commerce and network traffic analysis. The datasets in such domains are constantly updated with new data. Let us assume that at some point in time we have computed all frequent itemsets for such a dataset. Now, if the dataset is updated, then the set of frequent itemsets that we had previously computed will no longer be valid (some itemsets may still be frequent, but not all of them). A naïve approach to compute the new set of frequent itemsets would be to re-execute a traditional algorithm on the entire updated dataset, but this process is not efficient since it ignores the previously discovered knowledge, essentially replicating work that has already been performed, which can result in an explosion in the amount of computational and I/O resources required.

To address this problem several researchers have proposed incremental techniques [1]–[6]. Incremental algorithms essentially re-use previously mined information and try to combine this information with the fresh data to efficiently compute the new set of frequent itemsets. However, it can be the case that the size of the dataset is so large and rate at which it is being updated is so high that existing incremental algorithms are ineffective by themselves. Therefore, to mine such large and rapidly changing datasets, we must also rely on high-performance computing techniques. Furthermore, the database may be distributed over multiple sites, being updated at different rates at each site, which requires the use of distributed asynchronous data mining techniques.

Ideally, one would like to mine frequent itemsets interactively. To do so, one's query must be answered in as little time as possible. To achieve this, an algorithm for distributed, incremental mining must take into account the rate at which data arrives and the computational resources available at each site, since these two variables can vary from one site to the next. Additionally, one might like to know about the skewness of the distributed database, specifically how the supports of the frequent itemsets vary from site to site. For example, in the context of network intrusion detection, one might like to know how models of network traffic vary at different points in a network (e.g. routers). Therefore, the algorithm must be able to determine the high-contrast frequent itemsets.

In this article we propose an efficient parallel and distributed incremental approach for mining frequent itemsets on dynamic and distributed datasets. Our work here is an extension of that presented in [7]–[9]. Our main contributions are:

1) A parallel algorithm based on the ZIGZAG incremental approach, which is used to update the local model;
2) A distributed mining algorithm that minimizes the communication costs for mining over

a wide area network, which is used to update the global model;

3) Novel interactive extensions for computing high contrast frequent itemsets;
4) Experimentation and validation on both real and synthetic databases.

The rest of this article is organized as follows. Section II examines related work in the fields of distributed and dynamic data mining. In Section III we formally define the problem we address in this article. Section IV describes our incremental, parallel, and distributed algorithms. Section V presents experimental results that validate our claims. Concluding remarks are made in Section VI.

II. Background and Related Work

A. Parallel and Distributed Mining of Databases

Often, the size of a dataset or the rate at which data is inserted or removed is so large that existing sequential algorithms are ineffective. In these cases, parallel or distributed algorithms are necessary. In [10], Park and Kargupta give an overview of a wide variety of distributed data mining algorithms for association rule mining, classifier learning, collective data mining, and clustering, among others. In particular, there has been much research into parallel and distributed algorithms for mining association rules [11]–[15]. Zaki provides an overview of several of these methods and others in [16].

A common approach for mining distributed databases is the *centralized* one, in which all 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, some intermediate approaches have been proposed [11], [17]–[19]. In [17] three distributed mining approaches were proposed. The Count Distribution algorithm is a simple distributed implementation of Apriori [20]. 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 I/O 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 [11] 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 by more than one order of magnitude. PARMAXECLAT outperforms PARECLAT, but it searches only the *maximal* frequent itemsets, instead of all frequent itemsets.

These techniques are devised to scale up a given algorithm (e.g., APRIORI, ECLAT, etc.). 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 distributed over a geographically wide area. The FDM (Fast Distributed Mining) algorithm presented in [18] attempts to cut down on communication between sites. It does this by first having each site mine its local frequent itemsets and then exchanging these so that support counts can be taken across all sites to find the global frequent itemsets. In [21], Schuster and Wolff note that FDM does not scale well as the number of sites increases, and so propose the DISTRIBUTED DECISION MINER algorithm and several variations, which do not assume that each local frequent itemset is potentially a global frequent itemset. Additionally, the DISTRIBUTED DECISION MINER is efficient even in the presence of skewness in the data. Not only is it desirable to reduce the amount of communication involved in distributed mining, it is also desirable to adapt the algorithm to differences in the amount of computational and communication resources at each site. In [22], three strategies of distributed data mining are examined, based on what information is exchanged between sites. The three strategies are those that move results (MR), move models (MM), and move data (MD). The MD strategy is generally avoided, since it can be costly in terms of communication. The authors propose the Papyrus system, which makes use of all three strategies to different degrees, based on the values of a cost function and an error function that take into account the cost of transmitting data between nodes and the distribution of data over the nodes.

B. Mining Dynamic Databases

Some recent effort has been devoted to the problem of performing incremental data mining in dynamic databases. Parthasarathy and Ramakrishnan propose a parallel incremental method for performing two-dimensional discretization on a dynamic dataset in [23]. A method for incremental sequence mining named ISM is presented in [24]. It is based on the SPADE algorithm [25] for discovering frequent sequences. The incremental algorithm works by keeping track of the maximally frequent and minimally infrequent sequences in the original database. It combines this information with the incremental data to minimize the amount of the original database that need to be re-scanned. Incremental versions of the GSP [26]

and MFS [27] frequent sequence mining algorithms, respectively named GSP+ and MFS+ were presented in [28]. Unlike the ISM algorithm, GSP+ and MFS+ are able to handle both insertions and deletions, and are not limited to a vertical database layout. There has also been work done in incremental web usage mining [29].

More specifically, there has been work on the problem of incrementally mining frequent itemsets [1]–[6] in dynamic databases. Some of these algorithms cope with the problem of determining when to update the current model of frequent itemsets, while others update the model after an arbitrary number of updates [6]. To decide when to update, Lee and Cheung [4] propose the DELI algorithm, which uses statistical sampling methods to determine when the current model is outdated. A similar approach proposed by Ganti *et al* [3] monitors changes in the data stream. An efficient incremental algorithm, called ULI, was proposed by Thomas [5] *et al*. ULI strives to reduce the I/O requirements for updating the set of frequent itemsets by maintaining the previous frequent itemsets and the *negative border* [30] along with their support counts. The whole database is scanned just once, but the incremental database must be scanned as many times as the size of the longest frequent itemset.

III. PROBLEM DEFINITION

A. 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 set of transactions, where each transaction has a unique identifier (*tid*) and contains a set of items. A set of items is called an *itemset*. An itemset with exactly k items (where k is a nonnegative integer) is called a k-itemset. The *tidset* of an itemset C corresponds to the set of all transaction identifiers (*tids*) in which 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, denoted by $\sigma(C)$, 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 set of all maximal frequent itemsets is denoted as MFI.

Mining frequent itemsets is a difficult problem. Given m items, there are potentially 2^m frequent itemsets. Even modest databases contain thousands of items and hundreds of thousands of transactions, so the computation power, memory, and disk I/O requirements are very high. The high computational and space costs may be acceptable when $\mathcal{D}$ is static since the discovery is done off-line. Also, many mechanisms such as sampling, and memory and parallel computing techniques have been presented in the literature to reduce these costs.

B. Frequent Itemset Mining in Dynamic Datasets

Using $\mathcal{D}$ as a starting point, a set of new transactions d^+ is added and a set of old transactions d^- is removed, forming the dynamic dataset Δ (i.e., $\Delta = (\mathcal{D} \cup d^+) - d^-$) [1]. Let $s_\mathcal{D}$ be the minimum support used when mining $\mathcal{D}$, and F_D be the set of frequent itemsets obtained. Let Π be the information kept from the current mining that will be used in the next incremental mining operation. In our case, Π consists of $F_\mathcal{D}$ (i.e., all frequent itemsets, along with their support counts, in $\mathcal{D}$). An itemset C is frequent in Δ if $\sigma(C) \geq s_\Delta$. Note that an itemset C not frequent in $\mathcal{D}$, may become a frequent itemset in Δ. In this case, C is called an *emerged* itemset. If a frequent itemset in $\mathcal{D}$ remains frequent in Δ it is called a *retained* itemset.

C. Frequent Itemset Mining in Distributed Datasets

The dataset Δ can be divided into n partitions, $\delta_1, \delta_2, ..., \delta_n$. Each partition δ_i is assigned to a site S_i. We say that Δ 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 Δ and δ_i. We will call $C.sup$ the *global support count* of C, and $C.sup_i$ the *local support count* of C in δ_i. For a given minimum support s_Δ, C is *global frequent* if $C.sup \geq s_\Delta \times \mid \Delta \mid$; correspondingly, C is *local frequent* at δ_i, if $C.sup_i \geq s_\Delta \times \mid \delta_i \mid$. The set of all maximal global frequent itemsets is denoted as MFI_Δ, and the set of maximal local frequent itemsets at δ_i is denoted as MFI_{δ_i}. The task of mining frequent itemsets in distributed and dynamic datasets is to find F_Δ (i.e., all global frequent itemsets in Δ), with respect to a minimum support s_Δ and, more importantly, using Π and minimizing access to $\mathcal{D}$ (the original dataset) to enhance the algorithm's performance.

IV. Algorithms

In this section we describe the basic algorithms to solve the problems defined in the previous section. Our incremental algorithm, ZigZag, is presented in Section IV-A. We also present a parallel approach for mining the maximal frequent itemsets. In Section IV-B we describe our distributed and incremental algorithm. We prove that this algorithm generates an accurate global model of frequent itemsets, and also present an upper bound for the amount of communication necessary in the distributed and incremental mining operation. Finally, we describe approaches for interactively mining itemsets in a distributed setting.

[1] Since modification of an existing transaction may be handled as a deletion followed by a insertion we will assume, without loss of generality, that there are no transaction modifications

A. Parallel Incremental Algorithm

Almost all algorithms for mining frequent itemsets use the same procedure – first a set of candidates is generated, the 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 find MFI_Δ. Once MFI_Δ is found, it is straightforward to obtain all frequent itemsets (and their support counts) in a single dataset 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 MFI_Δ is much smaller than the number of candidates generated to directly find all frequent itemsets. The maximal frequent itemsets has been successfully used in several data mining tasks, including incremental mining of evolving datasets [6], [31].

1) The ZIGZAG *Algorithm:* In [6] an efficient incremental algorithm for mining evolving datasets, ZIGZAG, was proposed. The main idea is to incrementally compute MFI_Δ using previous knowledge Π. This avoids the generation and testing of many unnecessary candidates. Having MFI_Δ is sufficient to know which itemsets are frequent; their exact support can be obtained by examining d^+, d^- and using Π, or, where this is not possible, by examining Δ.

ZIGZAG employs a backtracking search to find MFI_Δ. Backtracking algorithms are useful for many combinatorial problems where the solution can be represented as a set $I = \{i_0, i_1, ...\}$, 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, $I_k = \{i_0, i_1, ..., i_{k-1}\}$, 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 $I_k = \{i_0, i_1, ..., i_{k-1}, y\}$ will not be considered by the backtracking algorithm. Each iteration of the algorithm tries to extend 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 backtracking search performs a depth-first traversal of the search space, as depicted in Figure 1. In this example the minimum support is 30%. The framed itemsets are the maximal frequent ones, while the cut itemsets are the infrequent ones.

The support computation employed by ZIGZAG is based on the associativity of itemsets, which is defined as follows. Let C be a k-itemset of items $C_1 \ldots C_k$, where $C_i \in I$. Let $\mathcal{L}(C)$ be its tidset and $\mid \mathcal{L}(C) \mid$ is the length of $\mathcal{L}(C)$ and thus the support count of C. According to [32], any itemset can be obtained by joining its atoms (individual items) and its support count can be obtained by intersecting the tidsets of its subsets. In the first step, ZIGZAG

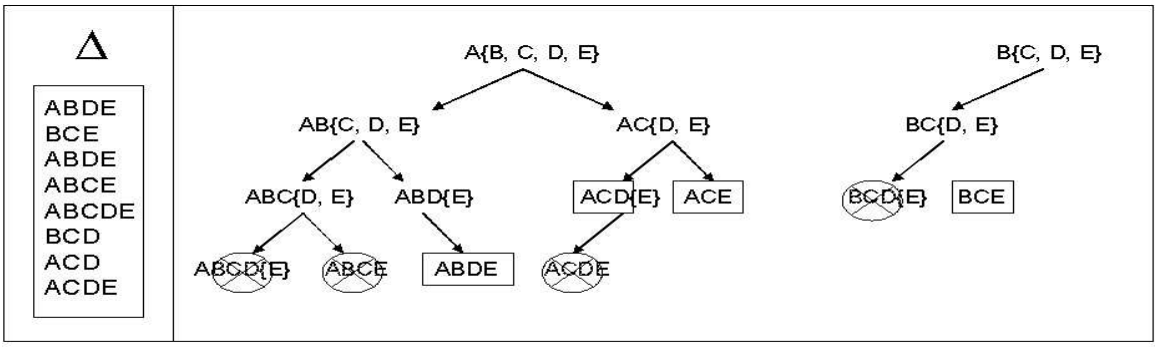

Fig. 1. Backtrack Trees for Items A and B on Δ

ZigZag(I_L, C_L, L)
for each $x \in C_L$ do
I_{L+1} = I $\cup$ {x}
P_{L+1} = $\{y : y \in C_L and\ y > x\}$
if $I_{L+1} \cup P_{L+1}$ has a superset in MFI
return
C_{L+1} = Extend(I_{L+1}, P_{L+1})
if $C_{L+1} == \emptyset$
if I_{L+1} has no superset in MFI
MFI = MFI $\cup I_{L+1}$
else
ZigZag(I_{L+1}, C_{L+1}, $L+1$)

Extend(I_{L+1}, P_{L+1})
$C = \emptyset$
for each $y \in P_{L+1}$ do
$y' = y$
if $I_{L+1} \cup \{y\}$ is a retained itemset
$\sigma_\Delta(y') = \sigma_D(I_{L+1} \cup \{y\}) + \sigma_{d+}(I_{L+1} \cup \{y\}) - \sigma_{d-}(I_{L+1} \cup \{y\})$
else
$\sigma_\Delta(y') = \sigma_D(I_{L+1} \cup \{y\})$
if $\sigma_\Delta(y') == s_\Delta$
$C = C \cup \{y'\}$
return

Fig. 2. Incremental Search for Maximal Frequent Itemsets

creates a tidset for each item in d^+, d^-, and Δ. The main goal of incrementally computing the support is to maximize the number of itemsets that have their support computed based just on d^+ and d^- (i.e., retained itemsets), since their support counts in $\mathcal{D}$ are already stored in $\mathcal{P}$. To perform a fast support computation, we first verify if the extension $I_{l+1} \cup \{y\}$ is a retained itemset. If so, its support can be computed by just using d^+, d^-, and $\mathcal{P}$, thereby enhancing the support computation process. All these procedures are described in Figure 2.

2) Parallel Incremental Search for Maximal Frequent Itemsets: We now consider the problem of parallelizing the search for maximal frequent itemsets in the shared memory paradigm (i. e., each processor has direct and equal access to all the system's memory). An efficient parallel search in this paradigm has two main issues: (1) minimizing synchronization (i. e., locks and barriers), and (2) improving data locality (i.e., maximizing access to local cache).

The main idea of our parallel approach is to assign distinct backtrack trees to distinct processors. Note from Figure 1 that the two issues mentioned above can be addressed by this approach. First, there is no dependence among the processors, because each backtrack tree

corresponds to a disjoint set of candidates. Since each processor can proceed independently there is no synchronization while searching for maximal frequent itemsets. Second, this approach is very efficient in achieving good data locality, since the support computation of an itemset is based on the intersection of the tidsets of the last two generated subsets. To achieve a suitable level of load-balancing, the trees are assigned to the processors by using the scheme of *bitonic partitioning* [33]. The bitonic scheme is a greedy algorithm, which first sort all the w_i (the work load due to tree i). w_i is calculated based on our ideas for estimating the number candidates using correlation measures, presented in [6]. Next it extracts the tree with maximum w_i, and assign it to processor 0. The next highest workload tree is assigned to processor 1 and so on.

When the *bag* is empty (i.e., all backtrack trees were processed), all maximal frequent itemsets have been found.

B. Distributed Incremental Algorithm

The MFI search employed by ZIGZAG is very efficient, but it can only be applied when the evolving dataset is centralized. Now we will explain how we can extend ZIGZAG for mining distributed datasets. We first present Lemma 1, which is the basic theoretical foundation of our approach.

a) Lemma 1 $-$ *A global frequent itemset must be local frequent in at least one partition.: Proof.* $-$ Let C be an itemset. If $C.sup_i < s_\Delta \times \mid \delta_i \mid$ for all $i = 1, ..., n$, then $C.sup < s_\Delta \times \mid \Delta \mid$ (since $C.sup = \sum_{i=1}^{n} C.sup_i$ and $\mid \Delta \mid = \sum_{i=1}^{n} \mid \delta_i \mid$), and C cannot be globally frequent. Therefore, if C is a global frequent itemset, it must be local frequent in some partition δ_i. $\square$

In the first step each site S_i independently performs a parallel and incremental search for MFI_{δ_i}, using ZIGZAG on its dataset δ_i. After all sites finish their searches, the result will be the set of all local MFIs, $\{\text{MFI}_{\delta_1}, \text{MFI}_{\delta_2}, \ldots, \text{MFI}_{\delta_i}\}$. This information is sufficient for determining all local frequent itemsets, and from Lemma 1, it is also sufficient for determining all global frequent itemsets. The second step starts after all local MFIs were found. Each site sends its local MFI to the other sites, and then they join all local MFIs. Now each site knows the set $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$, which is an upper bound for MFI_Δ.

In the third step each site independently performs a top down incremental enumeration of the potentially global frequent itemsets, as follows. Each itemset present in the upper bound $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$ is broken into k subsets of size $(k-1)$. This process iterates generating smaller subsets and incrementally 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).

b) Lemma 2 $-$ $\bigcup_{i=1}^{n}$ *MFI*$_{\delta_i}$ *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. If C is local frequent in some partition δ_l, so it must be determined by MFI_{δ_l}, and consequently by $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$. □

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 fourth and final step makes a reduction operation on the local support counts of each itemset, to verify which of them are globally frequent in Δ. 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 s_Δ, which constitutes the set of all global frequent itemsets, i.e., MFI_Δ

We illustrate all steps of the algorithm execution in Figure 3. The transactions of Δ (used in the example of Figure 1) were distributed in two datasets δ_1 and δ_2. The value of the minimum support is 50%. In the first step each site mines its local MFI. The result is MFI_{δ_1} = {ABDE, BCE}, and MFI_{δ_2} = {ACDE, BCD}. In the next step, all sites exchange their local MFIs, so that each one can compute the upper bound $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$, which is {ABDE, BCE, ACDE, BCD}. Now, each site computes the support count of each subset of each itemset in $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$. Some of the generated subsets at site S_i are not local frequents 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). In the last step the global frequent itemsets are found by aggregating (sum reduction operation) the local counts of each local frequent itemset.

The overall approach is as follows. At each site the local MFI is found using the appropriate version of the ZIGZAGaglorithm (the sequential version if the site is a single processor machine, or the parallel version if the site is a SMP machine or a cluster). Once the local MFIs have been found, the distributed procedure can be used across the different sites to find the global MFI.

1) 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. The upper bound calculation is based just on the local MFIs and on the size of the upper bound for MFI_Δ. We divide the upper bound construction into two steps. The first step is related to the local MFI exchange operation. Since each one of the n sites must send its MFI to the other sites, the first term is given by: $\sum_{i=1}^{n} \sum_{j=1}^{|MFI_{\delta_i}|} \mid C_{i,j} \mid$, where $\mid C_{i,j} \mid$ is the size of the j^{th} itemset of the local MFI of site S_i.

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 UB is $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$, and the term

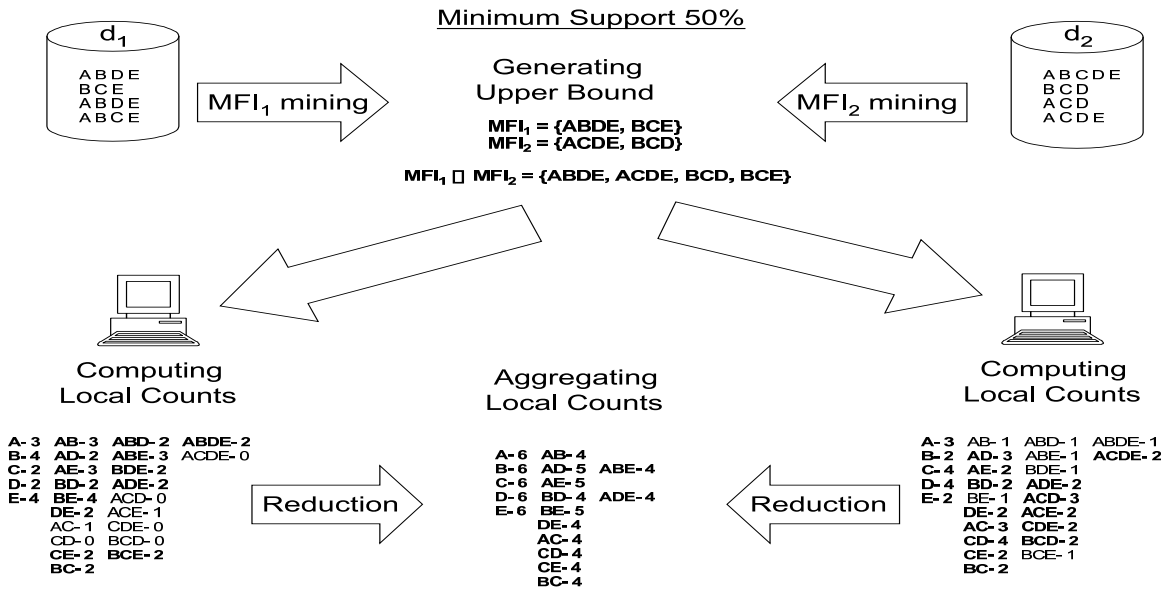

Fig. 3. Overall Process of Distributed Mining

$\sum_{i=1}^{|UB|}(2^{|C_j|}-1)$ 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 (when each itemset is subset of only one itemset in UB), the total amount of communication is given by: $(\sum_{i=1}^{n}\sum_{j=1}^{|MFI_{\delta_i}|} \mid C_{i,j} \mid + (n-1)\times\sum_{i=1}^{|UB|}(2^{|C_j|}-1)) \times 4$ bytes.

C. Interactivity Issues

1) Query Response Time: One of the goals of the distributed mining algorithm is to minimize response time to a query for the global frequent itemsets in an dynamic, distributed database. Since the database is dynamic, each site is incrementally updating its local frequent itemsets. The time it takes to update the local frequent itemsets is proportional to $B = \mid d^+ \mid + \mid d^- \mid$, that is to say, the size of a block of differences. We can view the updates to the database as a queue containing zero or more such blocks. If a query arrives while a block is being processed, there cannot be a response until the calculation of the local frequent itemsets is completed and used to find the global frequent itemsets. An obvious approach to reducing response time is to decrease the size of B. However, because of overhead, the time it takes to do two increments of size B is longer than the time it takes to do a single increment of size $2 \times B$. So there is a trade-off: If the size of B is increased, then the global frequent model MFI_Δ will be more up-to-date, since it incorporates a greater number of changes to the database, but the amount of time it takes to respond to the query will increase.

2) High-Contrast Frequent Itemsets: An important issue when mining distributed databases is to understand the differences between the databases. An effective way to understand such differences is to find the high-contrast frequent itemsets. The support counts of such

itemsets vary significantly across different databases. We use the well-established notion of entropy to detect how the support count of a given frequent itemset is distributed across the databases [34]. For a random variable, the entropy is a measure of the non-uniformity of its probability distribution. Let X be a global frequent itemset. The value $p_X(i) = \frac{X.sup_i}{X.sup}$ is the probability of occurrence of X in δ_i. $\sum_{i=1}^{n} p_X(i) = 1$, and $H(X) = -\sum_{i=1}^{n}(p_X(i) \times log(p_X(i)))$ is a measure of how the local support counts of X is distributed across the different databases. Note that $0 \leq H(X) \leq log(n)$, and so $0 \leq E(X) = \frac{log(n)-H(X)}{log(n)} \leq 1$. If $E(X)$ is greater than or equal to a given minimum entropy threshold, then X is classified as high-contrast frequent itemset.

D. Discussion

Mining the MFI to find the frequent subsets has several advantages, both in incremental and distributed mining. In this section we discuss some of these advantages.

All extant incremental mining algorithms make use of the negative border [4] to perform the incremental operation. The basic idea is to maintain the negative border up-to-date as the dataset is updated. As shown in [6], the size of the negative border is typically much larger than the size of MFI_Δ. So, updating the negative border requires many more candidates to be processed, incurring computational and I/O overhead. By updating MFI_Δ, we process fewer candidates than other approaches.

Almost all distributed algorithms for frequent itemset mining (CD [17], FDM [18], and DMA [19]) require a round of communication in every iteration of the algorithm. However, synchronization is implicit in communication, and therefore these algorithms suffer of communication overhead. Our approach overcomes the problem of communication overhead by making use of maximal frequent itemsets. Each site can independently search its local MFI, so no communication is needed during this search. After all local MFIs are found, only one round of communication is performed in order to build the upper bound. Again, each site can independently enumerates the local frequent itemsets, and after all local frequent itemsets are found, only one reduction operation is needed to find the global frequent itemsets. Therefore, by making use of maximal frequent itemsets, our distributed algorithm can asynchronously mine the frequent itemsets.

V. Experimental Evaluation

In this section we evaluate our algorithms and compare them to other approaches, and compare performance with respect to various datasets.

A. Experimental Setup

Our experimental evaluation of the algorithms presented in section IV-B was carried out on two clusters. The first cluster, *Cluster One*, consists of dual PENTIUM III 1GHz nodes

TABLE I
DATASET CHARACTERISTICS

Dataset	#Items	Avg. Length	#Transactions	Size
WPortal	3,183	4	7,786,137	428MB
WCup	5,271	8	7,618,927	645MB
T5I2D8000K	2,000	10	8,000,000	1,897MB

with 1GB of main memory Red Hat Linux 7.1. The second cluster, *Cluster Two*, consists of single PENTIUM III 933 MHz nodes with 512 MB of memory running Red Hat Linux 7.3. We further partitioned each cluster into two virtual clusters for a total of four clusters for some experiments. We assume that each database is distributed between the clusters, and that each node in the cluster has access to its cluster's portion of the database. Within each cluster, we have implemented the parallel program using the MPI message-passing library (MPICH over GM[2]), and for communication between clusters we use sockets.

We used several real and synthetic datasets for testing the performance of our algorithms. The *WCup* dataset comes from click-stream data from the official site of the 1998 World Soccer Cup. The *WPortal* dataset is derived from the click-stream data of a large Brazilian web portal. We scanned the logs and produced transaction files, 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) the request method is GET; (2) the request status is OK; and (3) the file type is HTML. A session starts with a request that satisfies the above properties, and ends when there has been no click from the client for 30 minutes. We also used a synthetic dataset (also available from IBM Almaden), which has been used as benchmarks for testing previous mining algorithms. This dataset mimics the transactions in a retailing environment [20].

Table I shows the characteristics of the real and synthetic datasets used in our evaluation. It shows the number of items, the average transaction length, the number of transactions, and the size of each dataset.

B. Parallel Incremental Algorithm Evaluation

The first experiment we conducted was to empirically verify the advantages of incremental and parallel mining within a single cluster. In this experiment we compared the execution time of the the distributed algorithms (non-incremental search, incremental search, parallel non-incremental search, and parallel incremental search). We used *Cluster One* only and varied

[2]www.myricom.com

the number of nodes (1 to 8), the number of processors (i.e. threads) per node (1 and 2), and the increment size (10% and 20% of the original dataset). In the incremental case, we first mine a determined number of transactions and then we incrementally mine the remaining transactions. For example, for increment sizes of 20%, we first mine 80% of the dataset and then we incrementally mine the remaining 20%.

Figure 4 shows the execution times obtained for different datasets, and parallel and incremental configurations. As we can see, better execution times are obtained when we combine both parallel and incremental approaches. Furthermore, when the parallel configuration is the same, the execution time is better for smaller increment sizes (since the dataset is smaller), but in some cases the parallel performance is greater than the incremental performance, and better results can be obtained by applying the parallel algorithm with larger increment sizes. This is exactly what happens in the experiments with the *WCup* and *WPortal* datasets. Not surprisingly, the gains from parallelizing the algorithm are greater than the gains from processing the data in an incremental fashion, especially given the number of processors. The algorithm using the parallel MFI search, applied to an increment size of 20% is more efficient than the algorithm with sequential MFI search, applied to an increment size of 10%, for any number of nodes.

1) Advantages of Parallel Mining: We also investigated the performance of our algorithm in experiments for evaluating the speedup of different parallel configurations. We used a fixed size dataset with increasing number of nodes. The dataset is divided into 1, 2, 4, and 8 partitions, according to the number of nodes employed. With this configuration we performed speedup experiments on 1, 2, 4, and 8 dual-processor nodes. In order to evaluate only the parallel performance, we used the parallel (two threads per node) non-incremental algorithm and varied the number of nodes. The speedup is in relation to the sequential non-incremental algorithm. Figure 5 shows the speedup numbers of our parallel algorithm. Our algorithm can achieve an efficiency of up to 80%, depending on the dataset, the number of processors, and the minimum support. Note that the speedup is inversely proportional to the minimum support. This is because for smaller minimum supports the MFI search becomes more complex, and consequently the parallel task becomes more relevant.

2) Advantages of Incremental Mining: We also investigated the performance of our algorithm in experiments for evaluating the speedup of different incremental configurations. In tis experiment, we first mined a fixed size dataset, and then we performed the incremental mining for different increment sizes (5% to 20%). In order to evaluate only the incremental performance, we used the incremental algorithm with sequential MFI search. We also varied the number of nodes, but the speedup was very similar for different number of nodes, so we show only the results regarding one node.

Figure 6 shows the speedup numbers of our incremental algorithm. Note that the speedup is in relation to re-mining the entire dataset. As is expected, the speed is inversely proportional

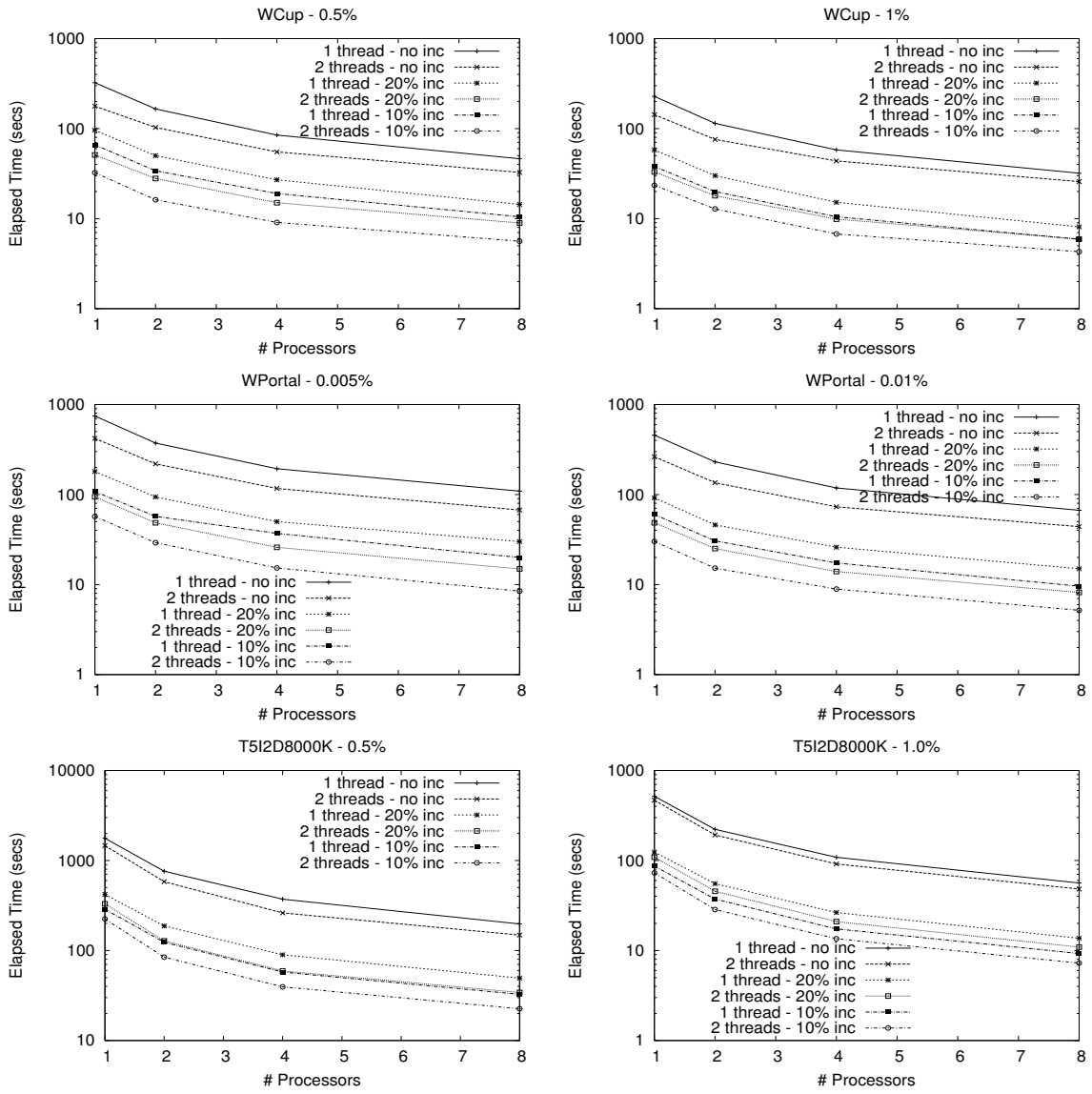

Fig. 4. Total Execution Times on different Datasets.

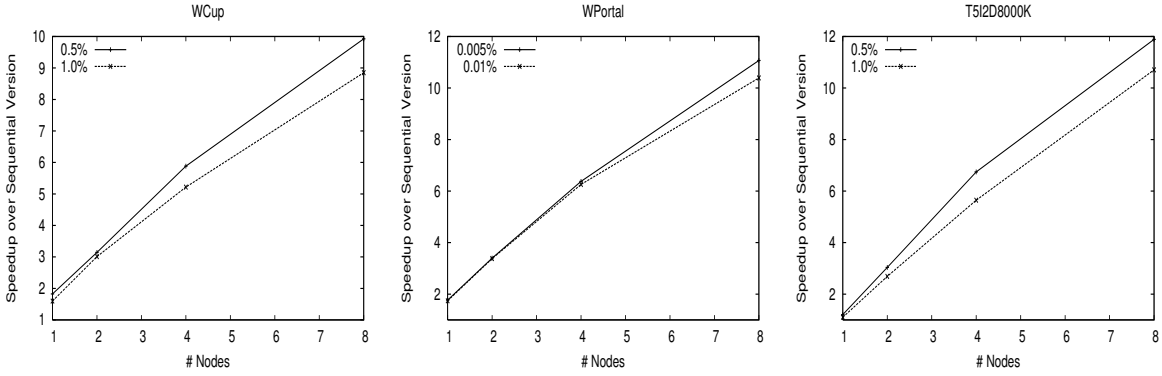

Fig. 5. Speedup of the Parallel Non-Incremental Algorithm

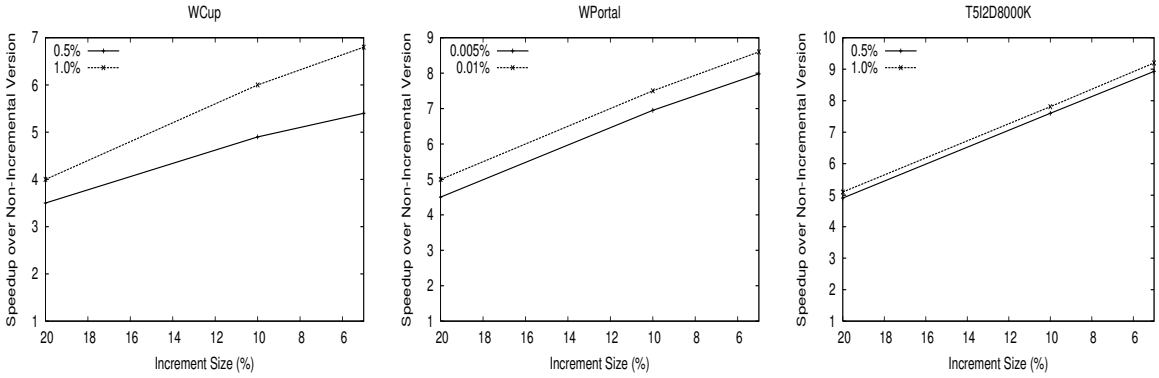

Fig. 6. Speedups on Different Incremental Configurations.

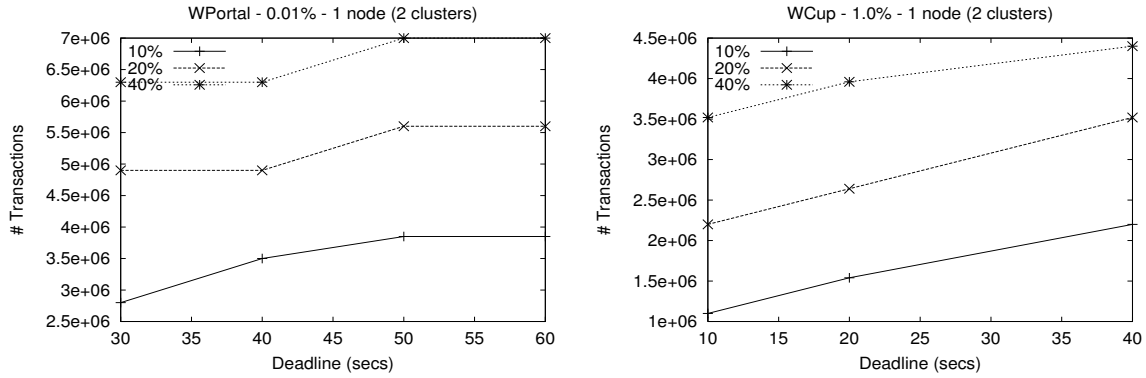

Fig. 7. Number of transactions processed.

to the size of the increment. This is because the size of the new data coming in is smaller. Also note that better speedups are achieved by greater minimum supports. We observed that, for the datasets used in this experiment, the proportion of retained itemsets (itemsets that are computed by examining only d^+ and Π) is larger for greater minimum supports. From Figure 5 and Figure 6, one can explicitly see the breakdown of the speedup provided by combining both parallel and incremental techniques. It is clear that parallel mining is a bigger component of the speedup than the incremental mining.

C. Distributed Incremental Algorithm Evaluation

We also performed several sets of experiments in a broader scenario involving several clusters. The first set involved finding the number of transactions that were processed and incorporated into the global model MFI_Δ when we varied certain parameters. The second set examined how the query response time was affected by the block size and the query arrival time.

1) Transactions Processed: The first experiment we conducted was to examine how the size of a block, the number of nodes used in each cluster, and the time at which a query arrives affects the amount of data used to build the global model $\text{MFI}_{\triangle}$. For the *WPortal* database we used a minimum support of 0.01% and for the *WCup* database we used a minimum support of 1.0%. The results all have similar trends, and two example cases can be seen in Figure 7. The X-axis represents the time elapsed from when the mining began until the query arrived (the deadline), and the Y-axis represents the number of transactions that are incorporated into the global model $\text{MFI}_{\triangle}$. The lines on the graph represent different values of the block size B, which is given here as percentages of the database on each cluster. The graphs show that if more time that elapses before the query arrives, the more data that can be incorporated into the model, which is to be expected. It also shows that as the block size decreases, fewer transactions can be processed before the query arrives. This is due to the fact there is more overhead involved in processing a large number of small blocks than there is in processing a small number of large blocks.

2) Query Response Time: In the next set of experiments we focused on the query response time, that is to say, the amount of time a user must wait before the global model is computed. For this experiment we varied the block size B (in these experiments we assume that each cluster use the same block size) and the time at which the query arrives. The results can be seen in Figure 8. The X-axis again represents the time at which the query arrives, and the Y-axis represents the time spent waiting for the global model to be computed. These graphs show that as B decreases, the time to wait for a response also decreases. However, the time at which a query arrives affects the waiting time in a seemingly random manner. This is because a query arrives at some random point during the processing of a block. The time remaining to compute the local frequent itemsets is therefore a random number. The graphs above show the query response time averaged over five runs, and the vertical bars represent the variance in the runs. This set of experiments, in conjunction with the previous set, clearly show the trade-off between block size and query response time: As the block size increases, the number of transactions processed also increases, but the response time increases as well. Figure 9 shows the same experiment when different block sizes across clusters are allowed. The basic idea is that smaller response times can be obtained by assigning larger blocks to the less powerful cluster. The local model in this cluster will be updated less frequently, but in response, smaller query response times can be obtained by this approach. We can prove this by comparing the results in Figure 9 against the respective result in Figure 8.

3) Communication: We also performed a set of experiments to analyze the communication overhead imposed by our algorithm. In particular we examined the number of bytes transferred between clusters when we varied the minimum support, the block size B, and the number of clusters involved in the computation. The results can be seen in figure 10. As is expected, as the minimum support decreases, the number of candidates will increase, and will therefore

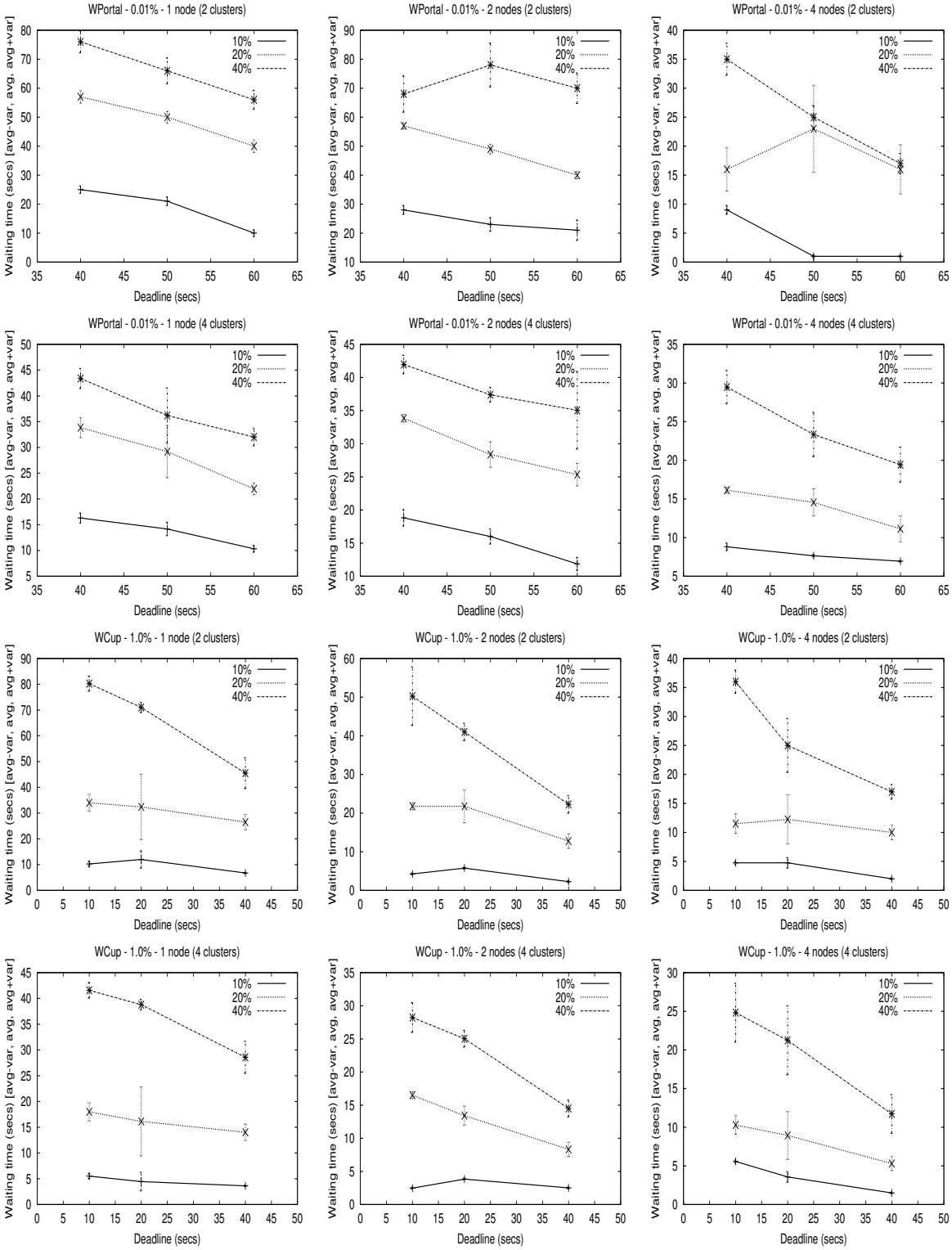

Fig. 8. Query Response Time using Equal Block Sizes.

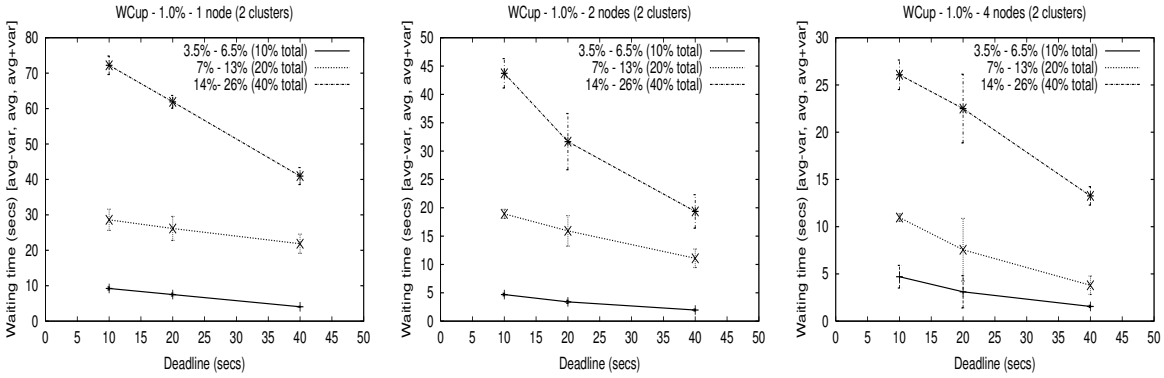

Fig. 9. Query Response Time using Different Block Sizes.

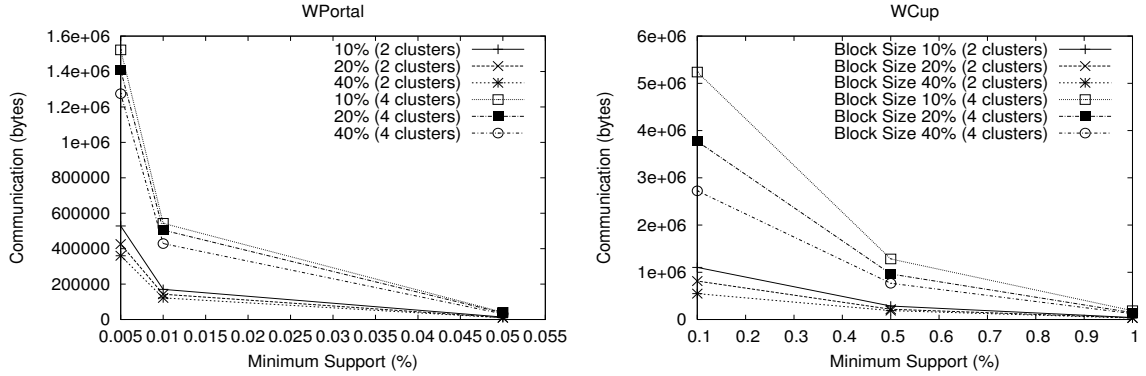

Fig. 10. Communication Overhead.

increase the number of bytes that must be transferred between the clusters, since our algorithm must exchange the support counts of every candidate processed. Also, as the block size increases, the amount of communication decreases. The reason is that for smaller block sizes the number of candidates processed tends to be greater (assuming the same minimum support). Finally, the amount of communication required increases when more clusters are involved in the process. However, the increasing factor is not linear because the data-skewness also increases when more clusters are involved, and so the number of candidates processed is increased as well.

4) High-Contrast Itemsets: The last set of experiments are regarding high-contrast frequent itemsets. We utilized three databases: *WPortal*, *WCup*, and also a highly-skewed synthetic database. The synthetic database was generated in the following way. We first generated four different synthetic databases: *T10I2D2000K*, *T10I4D2000K*, *T10I6D2000K* and *T10I8D2000K*. Next, each one of these databases was assigned to a different cluster. In this way we ensure that this distributed database contains highly-skewed data.

We varied three parameters: the minimum support, the number of clusters involved in the

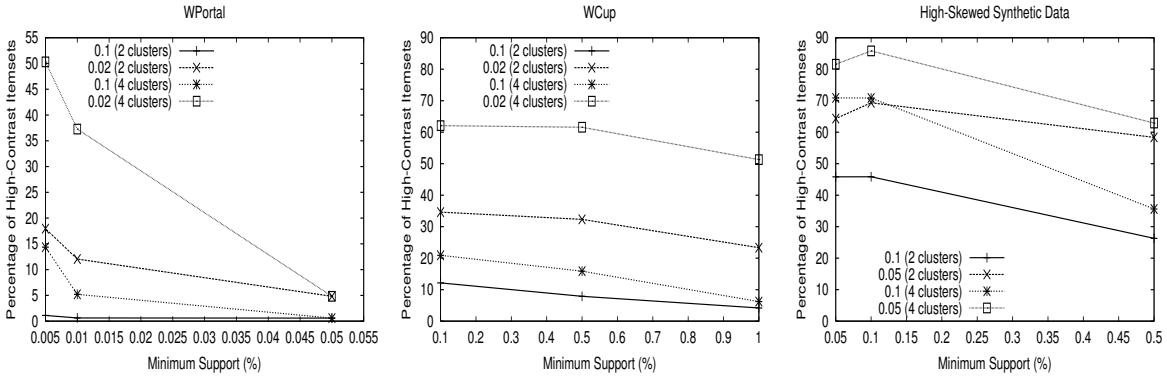

Fig. 11. High-Contrast Frequent Itemsets.

process, and the minimum entropy. Figure 11 shows the results obtained in each database. As we can observe, very different results were obtained from each database. The percentage of high-contrast frequent itemsets is interesting because it to some extent reveals the skewness of the database. From the experimental results, we know that the *WCup* database is more skewed than the *WPortal* database. Given the same support thresholds for these two databases, *WCup* will give a much higher percentage of high-contrast frequent itemsets. Usually the percentage of high-contrast frequent itemsets decreases as the minimum support threshold increases. This is quite understandable given that when the support threshold is low, there will be a large number of global frequent itemsets generated, and many of these itemsets become global frequent only because they have high local support at some site. In contrast, when the support threshold increases, it becomes harder for a local frequent itemset to become global frequent, which results in a decrease of high-contrast frequent itemsets. Accordingly, there is a higher proportion of high-contrast frequent itemsets in the former scenario. Furthermore, the more clusters over which the data distributed, the greater the possibility of skewness in the data. This is verified by the experimental data. It is interesting to notice that for the highly-skewed synthetic data, when the support threshold is incremented from 0.05 to 0.1, the percentage of high-contrast frequent itemsets did not increase as expected. We guess this can be attributed to the high skewness of the data. We surmise that for such data there exists some threshold for our claim to take effect. Taking our synthetic data as an example, the threshold value is around 0.1. Below this threshold, when we raised the support value, both of the high-contrast frequent itemsets and the global frequent itemsets decreased, but the loss of the high-contrast frequent itemsets was dominated by the loss of the global frequent itemsets, eventually leading to an increased percentage of high-contrast frequent itemsets.

VI. Conclusion and Future Work

In this article we have considered the problem of mining frequent itemsets on dynamic and distributed databases in different parallel and distributed environments. We presented an efficient distributed and parallel incremental algorithm to deal with this problem. In particular, we presented techniques to minimize the response time to a query for the global set of frequent itemsets, as well as to find high-contrast frequent itemsets. We believe that our approach is the first to deal with problems of dynamic and distributed data in a unified manner.

Our experimental results show that our algorithm results in execution time improvement of more than one order of magnitude when compared against a naive approach. The efficiency of our algorithm stems from the fact that it makes use of the MFI, reducing both the number of candidates processed and the amount of communication necessary. Our experiments in the distributed setting also examined the trade-offs involved in minimizing the query response time (whether to sacrifice query response time in order to incorporate more transactions in the model), the amount of data transferred between clusters, and how the distribution of the data affected the number of high-contrast frequent itemsets.

In the future, we plan to investigate the effectiveness of the WAVE algorithm [31] in parallel and distributed settings. The WAVE algorithm provides the ability to estimate support counts without scanning the entire database. Only when support counts can no longer be reliably estimated do we have examine the database. We would also like to explore sampling methods as a means of improving the query response time, and how to minimize query response time in wide-area networks, where communication latencies tend to be relatively large.

References

[1] D. Cheung, J. Han, V. Ng, and C. Y. Wong., "Maintenance of discovered association rules in large databases: An incremental updating technique," in *Proc. of the 12^{th} Int'l. Conf. on Data Engineering*, February 1996.

[2] D. Cheung, S. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," in *Proc. of the 5^{th} Int'l. Conf. on Database Systems for Advanced Applications*, April 1997, pp. 1–4.

[3] V. Ganti, J. Gehrke, and R. Ramakrishnan, "Demon: Mining and monitoring evolving data." in *Proc. of the 16^{th} Int'l Conf. on Data Engineering*, San Diego, USA, 2000, pp. 439–448.

[4] S. Lee and D. Cheung, "Maintenance of discovered association rules: When to update?" in *Research Issues on Data Mining and Knowledge Discovery*, 1997.

[5] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, "An efficient algorithm for the incremental updation of association rules," in *Proc. of the 3^{rd} ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, August 1997.

[6] A. Veloso, W. M. Jr., M. B. de Carvalho, B. Pôssas, S. Parthasarathy, and M. Zaki, "Mining frequent itemsets in evolving databases," in *Proc. of the 2^{nd} SIAM Int'l Conf. on Data Mining*, Arlington, USA, May 2002.

[7] M. E. Otey, A. Veloso, C. Wang, S. Parthasarathy, and W. Meira Jr., "Mining frequent itemsets in distributed and dynamic databases," in *IEEE International Conference on Data Mining*, 2003.

[8] 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 *International Workshop on High Performance Data Mining*, 2003.

[9] A. Veloso, M. E. Otey, S. Parthasarathy, and W. Meira Jr., "Parallel and distributed frequent itemset mining on dynamic datasets," in *International Conference On High Performance Computing*, 2003.

[10] B.-H. Park and H. Kargupta, "Distributed data mining: Algorithms, systems, and applications," in *Data Mining Handbook*, N. Ye, Ed., 2002.

[11] 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*, vol. 4, no. 1, pp. 343–373, December 1997.

[12] E.-H. Han, G. Karypis, , and V. Kumar, "Scalable parallel data mining for association rules," in *ACM SIGMOD Conf. Management of Data*, 1997.

[13] J. Park, M. Chen, , and P. S. Yu, "CACTUS - clustering categorical data using summaries," in *ACM Intl. Conf. Information and Knowledge Management*, 1995b.

[14] S. Parthasarathy, M. Zaki, M. Ogihara, and W. Li, "Parallel data mining for association rules on shared-memory systems," *Knowledge and Information Systems*, vol. 3, no. 1, pp. 1–29, 2001.

[15] M. J. Zaki, S. Parthasarathy, , and W. Li, "A localized algorithm for parallel association mining," in *Supercomputing'96*, 1997.

[16] M. J. Zaki, "Parallel and distributed association mining: A survey," *IEEE Concurrency*, vol. 7, no. 4, pp. 14–25, December 1997.

[17] R. Agrawal and J. Shafer, "Parallel mining of association rules," in *IEEE Trans. on Knowledge and Data Engg.*, vol. 8, 1996, pp. 962–969.

[18] D. Cheung, J. Han, V. Ng, A. Fu, , and Y. Fu, "A fast distributed algorithm for mining association rules," in *4th Intl. Conf. Parallel and Distributed Info. Systems*, 1996a.

[19] D. Cheung, V. Ng, A. Fu, , and Y. Fu, "Efficient mining of association rules in distributed databases," in *IEEE Trans. on Knowledge and Data Engg.*, vol. 8, 1996, pp. 911–922.

[20] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. of the 20^{th} Int'l Conf. on Very Large Databases*, SanTiago, Chile, June 1994.

[21] A. Schuster and R. Wolff, "Communication-efficient distributed mining of association rules," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. ACM Press, 2001, pp. 473–484.

[22] R. L. Grossman, S. M. Bailey, H. Sivakumar, and A. L. Turinsky, "Papyrus: A system for data mining over local and wide-area clusters and super-clusters," 1999. [Online]. Available: citeseer.nj.nec.com/408839.html

[23] S. Parthasarathy and A. Ramakrishnan, "Parallel incremental 2d discretization," in *to appear in IEEE International Conference on Parallel and Distributed Processing*, 2002.

[24] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas, "Incremental and interactive sequence mining," in *CIKM*, 1999, pp. 251–258. [Online]. Available: citeseer.nj.nec.com/article/parthasarathy99incremental.html

[25] Zaki, "Efficient enumeration of frequent sequences," in *CIKM: ACM CIKM International Conference on Information and Knowledge Management*. ACM, SIGIR, and SIGMIS, 1998. [Online]. Available: citeseer.nj.nec.com/zaki98efficient.html

[26] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, Eds., vol. 1057. Springer-Verlag, 25–29 1996, pp. 3–17. [Online]. Available: citeseer.nj.nec.com/article/srikant96mining.html

[27] M. Zhang, B. Kao, C. Yip, and D. Cheung, "A gsp-based efficient algorithm for mining frequent sequences," in *Proc. of IC-AI'001, Las Vegas, Nevada, USA*, June 2001. [Online]. Available: citeseer.nj.nec.com/zhang02gspbased.html

[28] M. Zhang, B. Kao, D. W.-L. Cheung, and C. L. Yip, "Efficient algorithms for incremental update of frequent sequences," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2002, pp. 186–197. [Online]. Available: citeseer.nj.nec.com/article/zhang02efficient.html

[29] F. Masseglia, P. Poncelet, and M. Teisseire, "Web usage mining: How to effciently manage new transactions and new clients." [Online]. Available: citeseer.nj.nec.com/400542.html

[30] H. Toivonen, "Sampling large databases for association rules," in *In Proc. 1996 Int. Conf. Very Large Data Bases*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. Morgan Kaufman, 09 1996, pp. 134–145. [Online]. Available: citeseer.nj.nec.com/toivonen96sampling.html

[31] A. Veloso, W. M. 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 6^{th} Int'l Conf. on Principles and Practices of Data Mining and Knowledge Discovery in Databases*, Helsinki, Finland, August 2002.

[32] K. Gouda and M. Zaki, "Efficiently mining maximal frequent itemsets," in *Proc. of the 1^{st} IEEE Int'l Conference on Data Mining*, San Jose, USA, November 2001.

[33] M. Cierniak, M. Zaki, and W. Li, "Compile-time scheduling algorithms for a heterogeneous network of workstations," in *The Computer Journal*, vol. 40, pp. 356–372.

[34] D. Cheung and Y. Xiao, "Effect of data skewness in parallel mining of association rules," in *Proc. of the 4^{th} Pacific-Asia Conference on Knowledge Discovery and Data Mining*, New York, USA, August 1998, pp. 48–60.