Agent Based Data Distribution for Parallel Association Rule Mining

Kamal Ali Albashiri

***Abstract*—** A Multi-Agent based approach to Data Mining using a Multi-Agent System (MADM) is described. The system comprises a collection of agents cooperating to address given Data Mining (DM) tasks. The exploration of the system is conducted by considering a specific parallel/distributed Association Rule Mining (ARM) scenario, namely data (vertical/horizontal) partitioning to achieve parallel/distributed ARM. To facilitate the partitioning a compressed set enumeration tree data structure (the T-tree) is used together with an associated ARM algorithm (Apriori-T). The aim of the scenario is to demonstrate that the MADM approach is capable of exploiting the benefits of parallel computing; particularly parallel query processing and parallel data accessing. Both of the data (vertical/horizontal) partitioning techniques are evaluated and compared. Comparison of the measures indicates that the data partitioning methods described are extremely effective in limiting the maximal memory requirements of the algorithms, while their execution time scale only slowly and linearly with increasing data dimensions.

***Keywords*—** Association Rule Mining, Multi-Agent Data Mining, Meta Mining, Frequent Itemsets, T-tree.

I. Introduction

THE advantages offered by Multi-Agent Systems (MAS) MADM can provide support to address a number of general data mining issues, such as:

1) The size of the data sets to be mined: Ultimately data miners wish to mine everything: text, images, video, multi-media as well as simple tabular data. DM techniques to mine tabular data sets are well established, however ever larger data sets, more complex data (images, video), and more sophisticated data formats (graphs, networks, trees, etc.) are required to be mined. The resources to process these data sets are significant; an MADM approach may therefore provide a solution.
2) Data security and protection: The legal and commercial issues associated with the security and protection of data are becoming of increasing significance in the context of data mining. The idea of sharing data for data mining by first compiling it into a single data warehouse is often not viable, or only viable if suitable preprocessing and annoimization is first undertaken. MADM provides a mechanism to support data protection.
3) Appropriateness of DM Algorithms: An interesting observation that can be drawn from the DM research conducted to date is that for many DM tasks (for example ARM) there is little evidence of a "best" algorithm suited to all data. Even when considering relatively straightforward tabular data, in the context of ARM, there is no single algorithm that produces the best (most representative) association rules in all cases. An agent-based process of negotiation/interaction, to agree upon the best result, seems desirable.
4) Resource intensive: Common feature of most DM tasks is that they are resource intensive and operate on large sets of data. Data sources measured in gigabytes or terabytes are quite common in DM. This has called for fast DM algorithms that can mine very large databases in a reasonable amount of time. However, despite the many algorithmic improvements proposed in many serial algorithms, the large size and dimensionality of many databases makes the DM of such databases too slow and too big to be processed using a single process. There is therefore a growing need to develop efficient parallel DM algorithms that can run on distributed systems.

There are several ways in which data distribution can occur, and these require different approaches to model construction, including:

• **Horizontal Data Distribution**. The most straight forward form of distribution is horizontal partitioning, in which different records are collected at different sites, but each record contains all of the attributes for the object it describes. This is the most common and natural way in which data may be distributed. For example, a multinational company deals with customers in several countries, collecting data about different customers in each country. It may want to understand its customers worldwide in order to construct a global advertising campaign.

• **Vertical Data Distribution**. The second form of distribution is vertical partitioning, in which different attributes of the same set of records are collected at different sites. Each site collects the values of one or more attributes for each record and so, in a sense, each site has a different view of the data. For example, a credit-card company may collect data about transactions by the same customer in different countries and may want to treat the transactions in different countries as different aspects of the customers total card usage. Vertically partitioned data is still rare, but it is becoming more common and important [9].

This paper addresses a generic MADM scenario, that of distributed/parallel DM. This scenario assumes an end user

K.A. Albashiri, is a university lecturer at Al-Ghabel Al-Gharbi University, Gharin, Libya. He is with the Department of Data Analysis, Faculty of Accounting, Maydan Aljazer, P.O.Box 1531, Tripoli, Libya.(e-mail: elbashiri0@yahoo.com)

who owns a large data set and wishes to obtain DM results but lacks the required resources (i.e. processors and memory). The data set is partitioned into horizontal or vertical partitions that can be distributed among a number of processors (agents) and independently processed, to identify local itemsets, on each process.

In the exploration of the applicability of MADM to parallel/distributed ARM, the two data partitioning approaches, based on the Apriori algorithm, are described and their performance evaluated as indicated above. Recall that DATA-HS (Horizontal Segmentation) makes use of a horizontal partitioning of the data. The data is apportioned amongst a number of data agents, typically by horizontally segmenting the dataset into sets of records.

DATA-VP makes use of a vertical partitioning approach to distributing the input dataset over the available number of DM (worker) agents. To facilitate the vertical data partitioning the tree data structure, described in [7], is again used together with the Apriori-T ARM algorithm [6]. Using both approaches each partition can be mined in isolation, while at the same time taking into account the possibility of the existence of frequent itemsets dispersed across two or more partitions. In the first approach, DATA-HS, the scenario complements the meta ARM scenario described in Albashiri *et al*. [3].

The rest of the paper is organized as follows. Section II provides an overview of the field of MADM. Data partitioning is introduced in Section III. Data partitioning may be achieved in either a horizontal or vertical manner. A brief note on the data structures used by the ARM algorithms is then presented in Section IV. Before describing the data partitioning approaches the Apriori-T algorithm is briefly described in Section V. In Section VI a parallel/distributed task with Data Horizontal Segmentation (DATA-HS) algorithm is described. the nature of the agent communication protocols is given in Section VII. The parallel/distributed task with Data Vertical Partitioning (DATA-VP) algorithm (which is founded on Apriori-T) is then described in Section VIII. The DATA-VP MADM task architecture and network configuration is presented in Section IX. Experimentation and Analysis, comparing the operations of DATA-HS and DATA-VP, is then presented in Section X. Discussion of how this scenario addresses the goal of this paper is presented in Section XI. Finally a conclusion is given in Section XII.

II. Background and Related Work

This section briefly presents a review of the current research relating to Multi-Agent Data Mining (MADM). It provides an overview of the theoretical background of the research discipline, identifying the approaches adopted, and discusses the benefits and challenges posed.

During the last two decades, our ability to collect and store data has significantly outpaced our ability to analyze, summarize and extract "knowledge" from this data. The phrase Knowledge Discovery in Databases (KDD) denotes the complex process of identifying valid, novel, potentially useful and ultimately understandable patterns in data [15]. DM refers to a particular step in the KDD process. It consists of particular algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (models) over the data.

A considerable number of algorithms have been developed to perform DM tasks, from many fields of science [16]. Typical DM tasks are classification (the generation of classifiers which can be used to assign each record of a database to one of a predefined set of classes), clustering (finding groups of database records that are similar according to some user defined metrics) or ARM (determining implication rules for a subset of database record attributes).

Agents and multi-agent systems are an emergent technology that is expected to have a significant impact in realizing the vision of a global and information rich services network to support dynamic discovery and interaction of digital enterprises. Significant work on multi-agent systems has already been done for more than a decade since agents were first claimed to be the next breakthrough in software development, resulting in powerful multi-agent platforms and innovative e-business applications.

Multi-agent Data Mining (MADM) is concerned with the use of agent and MAS to perform DM activities. MAS has some particular advantages to offer with respect to Knowledge Discovery in Data (KDD), and particularly data mining, in the context of sharing resources and expertise.

KDD has evolved to become a well established technology that has many commercial applications. Research work in these fields continues to develop ideas, generate new algorithms and modify/extend existing algorithms. A diverse body of work therefore exists. KDD research groups and commercial enterprises, are prepared (at least to some extent) to share their expertise. In addition, many KDD research groups have made software freely available for download1. This all serves to promote and enhance the current "state of the art" in KDD. However, although the free availability of data mining software is of a considerable benefit to the KDD community, it still require users to have some programming knowledge — this means that for many potential end users the use of such free software is not a viable option. It is proposed in this paper that this disadvantage can be addressed by using the MAS mode of operation.

An additional advantages offered by MAS, in the context of data mining, is that of privacy and (to an extent) security. By its nature data mining is often applied to sensitive data. MAS allows data to be mined remotely. Similarly, with respect to data mining algorithms, MAS can make use of algorithms with necessitating their transfer to users, thus contributing to the preservation of intellectual property rights.

Several systems have been developed for MADM. These systems can be categorized, according to their strategy of learning, into three types:

1) Central-learning, when all the data can be gathered at a central site and a single model built [18], [21].
2) Meta-learning, is the process of automatic induction of correlations between tasks and solving strategies, based on a domain characterization [19], [17], [23].
3) Hybrid-learning is a technique that combines local and remote learning for model building [20], [22].

The most popular task of DM is to find patterns in data that show associations between domain elements. This is generally focused on transactional data, such as a database of purchases at a store. This task is known as Association Rule Mining (ARM), and was first introduced in Agrawal *et al*. [2]. Association Rules (ARs) identify collections of data attributes that are statistically related in the underlying data.

III. Segmentation And Partitioning

Notwithstanding the extensive work that has been done in the field of ARM, there still remains a need for the development of faster algorithms and alternative heuristics to increase their computational efficiency. Because of the inherent intractability of the fundamental problem, much research effort has been directed at parallel ARM to decrease overall processing times (see [8], [11], [12], [13]), and distributed ARM to support the mining of datasets distributed over a network [4]. The main challenges associated with parallel DM include:

- Minimizing I/O.
- Minimizing synchronization and communication.
- Effective load balancing.
- Effective data layout (horizontal vs. vertical).
- Good data decomposition.
- Minimizing/avoiding duplication of work.

To allow the data to be mined using a number of cooperating agents the most obvious approach is to allocate different subsets of the data to each agent. There are essentially two fundamental approaches to partitioning/segmenting the data:

1) Horizontal segmentation where the data is divided according to row number.
2) Vertical partitioning where the data is divided according to column number.

Note that in this paper the term partitioning is used to indicate vertical subdivision of data, and segmentation to indicate horizontal subdivision of data.

Horizontal segmentation, is in general more straightforward. Assuming a uniform/ homogeneous dataset it is sufficient to divide the number of records by the number of available agents and allocate each resulting segment accordingly.

The most natural method to vertically partition a dataset is to divide the number of columns by the number of available agents so each is allocated an equal number of columns.

Many parallel DM algorithms have been developed based on the Apriori algorithm or variations of the Apriori algorithm. The most common parallel methods are [2], [8]:

- Count Distribution. This method follows a data-parallel strategy and statically partitions the database into horizontal partitions that are independently scanned for the local counts of all candidate itemsets on each process. At the end of each iteration, the local counts are summed across all processes to form the global counts so that frequent itemsets can be identified.
- Data Distribution. The Data Distribution method attempts to utilize the aggregate main memory of parallel machines by partitioning both the database and the candidate itemsets. Since each candidate itemset is counted by only one process, all processes have to exchange database partitions during each iteration in order for each process to get the global counts of the assigned candidate itemsets.
- Candidate Distribution. The Candidate Distribution method also partitions candidate itemsets but selectively replicates, instead of "partitioned-exchanging" the database transactions, so that each process can proceed independently.

Experiments show that the Count Distribution method exhibits better performance and scalability than the other two methods [2]. The steps for the Count Distribution method may be generalized as follows (for distributed-memory multiprocessors):

1) Divide the database evenly into horizontal partitions among all processes.
2) Each process scans its local database partition to collect the local count of each item.
3) All processes exchange and sum up the local counts to get the global counts of all items and find frequent 1-itemsets.
4) Set level k = 2.
5) All processes generate candidate k-itemsets from the mined frequent (k-1)-itemsets.
6) Each process scans its local database partition to collect the local count of each candidate k-itemset.
7) All processes exchange and sum up the local counts into the global counts of all candidate k-itemsets and find frequent k-itemsets among them.
8) Repeat (5) - (8) with k = k + 1 until no more frequent itemsets are found.

In the following sections two MADM tasks, using both vertical partitioning and horizontal segmentation, are introduced. These tasks were implemented using a task wrapper, so that they could be incorporated into the system as task agents.

IV. Note on P and T Trees

The Meta ARM algorithms described here make use of two data structures, namely P-trees and T-trees. The nature of these structures is described in detail in [3]; however, for completeness a brief overview is presented here.

The P-tree (Partial support tree) is a set enumeration tree style structure with two important differences: (i) more than one item may be stored at any individual node, and (ii) the tree includes partial support counts. The structure is sued to store a compressed version of the raw data set with partial support counts obtained during the reading of the input data. The best way of describing the P-tree is through an example such as that given in Fig. 1.

In the figure the data set given on the left is stored in the P-tree on the right. The advantages offered by the P-tree are of particular benefit if the raw data set contains many common leading sub-strings (prefixes). The number of such sub-strings can be increased if the data is ordered according to the frequency of the 1-itemsets contained in the raw data. The likelihood of common leading sub-strings also increases with the number of records in the raw data.

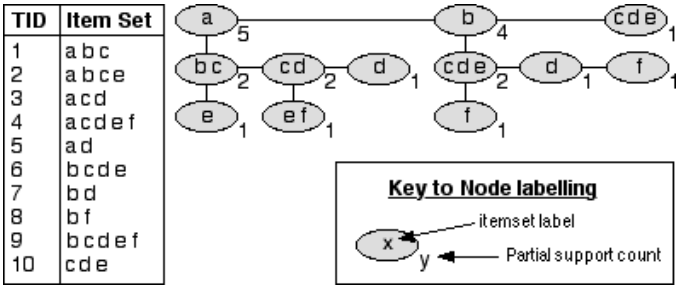


Fig. 1. P-tree example

The T-tree (Total support tree) is a "reverse" set enumeration tree structure that inter-leaves node records with arrays. It is used to store frequent item sets, in a compressed form, identified by processing the P-tree. An example, generated from the P-tree given Fig. 1, is presented in Fig. 2. From the figure it can be seen that the top level comprises an array of references to node structures that hold the support count and reference to the next level (providing such a level exists). Indexes equate to itemset numbers although for ease of understanding in the figure letters have been used instead of numbers.

The structure can be though of as a "reverse" set enumeration tree because child nodes only contain itemsets that are lexicographically before the parent itemsets. This offers the advantage that less array storage is required (especially if the data is ordered according to the frequency of individual items).

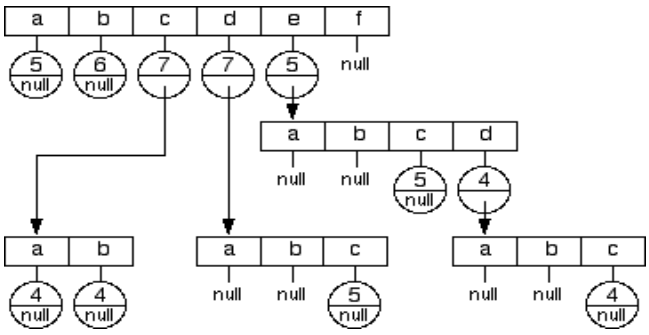


Fig. 2. T-tree example (support = 35%)

The T-tree is generated using an algorithm called Total From Partial (TFP) which is also described in [7]. The TFP algorithm is essentially an Apriori style algorithm that proceeds in a level by level manner. At each level the P-tree is processed to generate appropriate support counts. Note that on completion of the TFP algorithm the T-tree contains details of all the supported itemsets, in a manner that provides for fast look up during AR generation, but no information about unsupported sets (other than that they are not supported). Referring to Fig. 2 unsupported sets are indicated by a null reference.

V. The Apriori-T Algorithm

The Apriori-T (Apriori Total) algorithm is an Association Rule Mining (ARM) algorithm [7] that combines the classic Apriori ARM algorithm with the T-tree data structure. As each level is processed, candidates are added as a new level of the T-tree, their support is counted, and those that do not reach the required support threshold pruned. When the algorithm terminates, the T-tree contains only frequent itemsets. The Apriori-T algorithm was developed as part of the more sophisticated ARM algorithm The Apriori-TFP. The Apriori and Apriori-TFP algorithms are described in [7].

At each level, new candidate itemsets of size k are generated from identified frequent k-1 itemsets, using the downward closure property of itemsets, which in turn may necessitate the inspection of neighboring branches in the T-tree to determine if a particular k-1 subset is supported. This process is referred to as X-checking. Note that X-checking adds a computational overhead; offset against the additional effort required to establish whether a candidate k itemset, all of whose k-1 itemsets may not necessarily be supported, is or is not a frequent itemset.

The number of candidate nodes generated during the construction of a T-tree, and consequently the computational effort required, is very much dependent on the distribution of columns within the input data. Best results are produced by ordering the dataset, according to the support counts for the 1-itemsets, so that the most frequent 1-itemsets occur first [5].

VI. The Parallel/Distributed Task With Horizontal Segmentation (Data-Hs) Algorithm

The Data Horizontal Segmentation (DATA-HS) algorithm uses horizontal segmentation, dividing the dataset into segments each containing an equal number of records. ARM in this case involves the generation of a number of T-trees, holding frequent itemsets, one for each segment; and then merging these T-trees to create one global T-tree.

The most significant issue when combining groups of previously identified frequent sets is that wherever an itemset is frequent in a data source A but not in a data source B a check for any contribution from data source B is required (so as to obtain a global support count). The challenge is thus to combine the results from N different data sources in the most computationally efficient manner. This in turn is influenced predominantly by the magnitude (in terms of data size) of returns to the source data that are required.

The term *meta mining* is defined, in this paper , as the process of combining the individually obtained results of *N* applications of a DM activity. The motivation behind the scenario is that data relevant to a particular DM application may be owned and maintained by different, geographically dispersed, organizations. One approach to addressing the meta mining problem is to adopt a distributed approach. The meta mining scenario considered here is a meta Association Rule Mining (meta ARM) scenario where the results of *N* ARM operations, by *N* agents, are brought together.

A. Dynamic Behavior of System for Meta ARM operations

The meta ARM illustration described here was used to identify the most efficient Meta ARM agent process given a number of alternatives. The first algorithm was a bench mark algorithm, against which other Meta ARM algorithms were compared.

Four comparison meta ARM algorithms were constructed (Apriori, Brute Force, Hybrid 1 and Hybrid 2). Full details of the algorithms can be found in [3]. In each case it was assumed that each data source would produce a set of frequent sets, using some ARM algorithm, with the results stored in a common data structure. These data structures would then be merged in some manner through a process of agent collaboration. Each of the Meta ARM algorithms made use of a Return To Data (RTD) lists, one per data set, to contain lists of itemsets whose support was not included in the original ARM operation and for which the count was to be obtained by a return to the raw data held at a data agent. The RTD lists comprised zero, one or more tuples of the form < *I*, *sup* >, where *I* is an item set for which a count is required and *sup* is the desired count. RTD lists are constructed as a meta ARM algorithm progresses. During RTD list construction the *sup* value will be 0, it is not until the RTD list is processed that actual values are assigned to *sup*. The processing of RTD lists may occur during, and/or at the end of, the meta ARM process depending on the nature of the meta ARM algorithm used.

The meta ARM scenario comprises a set of *N* data agents and *N* +1 DM agents: *N* ARM agents and one meta ARM agent. Note that each ARM agent could have a different ARM algorithm associated with it, however a common data structure was assumed to facilitate data interchange. The common data structure used was a T-tree, a set enumeration tree structure for storing item sets.

Once generated the *N* local T-trees were passed to the Meta ARM agent which created a global T-tree. During the global T-tree generation process the Meta ARM agent interacted with the various ARM agents in the form of the exchange of RTD lists.

In this paper the the Apriori Meta ARM algorithm is used. For the Apriori Meta ARM algorithm, it was assumed that each data source would produce a set of frequent sets stored in a T-tree. These T-trees would then be merged in some manner.

K = 1
Generate candidate K-itemsets
Start Loop
if (K-itemsets == null break)
Add supports for level K from N T-trees or add to RTD list
Prune K-itemsets according to support threshold
K = K+1
Generate K-itemsets
End Loop

Table 1: Apriori Meta ARM algorithm

The Apriori Meta ARM algorithm described briefly below makes use of return to data (RTD) lists, one per data set, to contain lists of itemsets whose support was not included in the current T-tree and for which the count is to be obtained by a return to the raw data. RTD lists comprise zero, one or more tuples of the form < I, sup >, where I is an item set for which a count is required and sup is the desired count. RTD lists are constructed as the algorithm progresses. During RTD list construction the sup value will be 0, it is not until the RTD list is processed that actual values are assigned to sup. The processing of RTD lists may occur during, and/or at the end of, the Meta ARM process depending on the nature of the algorithm. If the RTD lists are not processed until the end of the merge phase. This means that many itemsets may be included in the merged T-tree sofar and/or the RTD lists that are in fact not supported.

The objective of the Aprori Meta ARM algorithm is to identify such unsupported itemsets much earlier on in the process. The algorithm proceeds in a similar manner to the standard Apriori algorithm as shown in Table 1. Note that items are added to the RTD list for data source n if a candidate itemset is not included in T-tree n. At the end of the merge phase the final merged T-tree is then pruned in phase three to remove any unsupported frequent sets according to the user supplied support threshold (expressed as a percentage of the total number of records under consideration). Further details of this Meta ARM process can be found in Albashiri *et al*. [3].

Assuming that a data agent representing the large dataset has been launched by a user, the DATA-HS MADM algorithm comprises the following steps:

1) User agent requests the task agent to horizontally segment the dataset according to the total number of segments required.
2) The task agent assigns and sends each data segment to an interested data agent; if none exist then the task agent launches new data agents.
3) Then a meta ARM task is called to obtain the Association Rules (ARs) as described in [3].

VII. Agent Communication

Agents are identified by name; to communicate to one another, an agent sender sends a message to another agent receiver by specifying the message and receiver name. In JADE applications, the agents communicate by sending messages which are objects and identify each agent by a predefined constant name and a variable instance local name. Any request for container location is a REQUEST message and any result to a REQUEST message is a INFORM message according to ACL specifications. Notice that each message has an associated Result, Query or Location object. JADE (Java Agent Development Environment) [14] is a multi-agent platform which this system is implemented in.

The system initially starts up with the two central JADE agents. When a user wishes to make its data available for possible data mining tasks, the user starts a data agent which in turn publish its name and description with the DF agent. In the context of ARM generation task, each DM agent could apply a different data mining algorithm to the data to produce its local T-tree. The frequent itemsets from each DM agent is collected by the task agent. Then the task agent merge the T-trees to generate one global T-tree. Once the T-tree is, association rules are generated and shown to the user through the user interface agent.

VIII. The Parallel/Distributed Task With Vertical Partitioning (Data-VP) Algorithm

The second algorithm considered in the exploration of the applicability of MADM to parallel/distributed ARM is the Data Vertical Partitioning (DATA-VP). The DATA-VP algorithm commences by distributing the input dataset over the available number of workers (DM agents) using a vertical partitioning strategy. Initially the set of single attributes (columns) is split equally between the available workers so that an allocationItemSet (a sequence of single attributes) is defined for each DM agent in terms of a startColNum and endColNum:

allocationItemSet = {n|startColNum < n endColNum}

Each DM agent will have its own allocationItemSet which is then used to determine the subset of the input dataset to be considered by the DM agent.

Using its allocationItemSet the task agent will partition the data among workers (DM agents) as follows:

1) Remove all records in the input dataset that do not intersect with the allocationItemSet.
2) From the remaining records remove those attributes whose column number is greater than endColNum. Attributes whose identifiers are less than startColNum cannot be removed because these may still need to be included in the subtree counted by the DM agent.
3) Send the allocated data partition to the corresponding DM agent.

The input dataset distribution procedure, given an allocationItemSet, can be summarized as follows:

```
 records  input data
if (record ∩
allocationItemSet ≡ true)
record = {n|n
n ≤ endColNum}  else
delete record
```

TID	*ItemSet*
1	*acf*
2	*b*
3	*ace*
4	*ad*
5	*ae*
6	*abc*
7	*d*
8	*ab*
9	*c*
10	*abd*

Table 2: Dataset Example

As an example, the ordered data set in Table 2 has items with 6 attributes, a, b, c, d, e and f. Assuming three worker agents are participating, the above partitioning process will result in three dataset partitions, with allocationItemSets {a, b}, {c, d} and {e, f}. Application of the above algorithm will create partitions as follows (but note that the empty sets, here shown for clarity, will in fact not be included in the partitions):

Partition 1 (a to b): {{*a*}, {*b*}, {*a*}, {*b*}, {*a*}, {*a, b*}, {}, {*a, b*}, {}, {*a, b*}}

Partition 2 (c to d): {{a, c}, {}, {a, c}, {b, d}, {}, {a, b, c}, {d}, {}, {c},{a, b, d}}

Partition 3 (e to f): {{*a, c, f*}, {}, {*a, c, e*}, {}, {*a, e*}, {}, {}, {}, {}}

Once partitioning is complete each partition can be mined, using the Apriori-T algorithm, in isolation while at the same time taking into account the possibility of the existence of frequent itemsets dispersed across two or more partitions.

Fig. 3 shows the resulting sub T-trees assuming all combinations represented by each partition are supported. Note that because the input dataset is ordered according to the frequency of 1-itemsets the size of the individual partitioned sets does not necessarily increase as the endColNum approaches N (the number of columns in the input dataset); in the later partitions, the lower frequency leads to more records being eliminated. Thus the computational effort required to process each partition is roughly balanced.

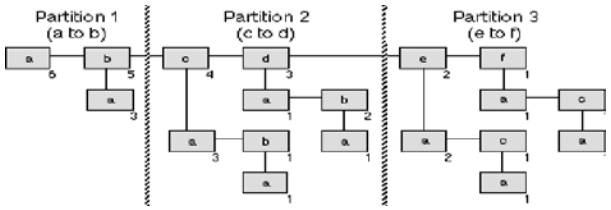


Fig. 3: Vertical Partitioning of a T-tree Example [6]

The DATA-VP MADM task can thus be summarized as follows:

1) A task agent starts a number of workers (DM agents); these will be referred to as *workers*.
2) The task agent determines the division of allocationItemSet according to the total number of available workers (agents) and transmits this information to them.
3) The task agent transmits the allocated partition of the data (calculated as described above) to each worker.
4) Each worker then generates a T-tree for its allocated partition (a sub T-tree of the final T-tree).
5) On completion each DM (worker) agent transmits its partition of the T-tree to the task agent which are then merged into a single global T-tree (the final T-tree ready for the next stage in the ARM process, rule generation).

The local T-tree generation process begins with a top-level "tree" comprising only those 1-itemsets included in each worker (DM agent) allocationItemSet.

The DM agent will then generate the candidate 2-itemsets that belong in its sub (local) T-tree. These will comprise all the possible pairings between each element in the allocationItemSet and the lexicographically preceding attributes of those elements (see Fig. 3). The support values for the candidate 2-itemsets are then determined and the sets pruned to leave only frequent 2-itemsets. Candidate sets for the third level are then generated.

Again, no attributes from succeeding allocationItemSet are considered, but the possible candidates will, in general, have subsets which are contained in preceding allocationItemSet and which, therefore, are being counted by some other DM agents. To avoid the overhead involved in the X-checking

process, described in Section 4, which in this case would require message-passing between the DM agents concerned, X-checking does not take place. Instead, the DM agent will generate its candidates assuming, where necessary, that any subsets outside its local T-tree are frequent.

IX. DATA-VP TASK ARCHITECTURE AND NETWORK CONFIGURATION

The DATA-VP task architecture shown in Fig. 4 assumes the availability of at least one worker (DM agent), preferably more. Fig. 4 shows the assumed distribution of agents and shared data across the network. The figure also shows the house-keeping JADE agents (AMS and DF) through which agents find each other.

A. Messaging

Parallel/distributed ARM tends to entail much exchange of data messaging as the task proceeds. Messaging represents a significant computational overhead, in some cases outweighing any other advantage gained. Usually the number of messages sent and the size of the content of the message are significant factors affecting performance. It is therefore expedient, in the context of the techniques described here, to minimize the number of messages that are required to be sent as well as their size.

The technique described here is One-to-Many approach, where only the task agent can send/receive messages to/from DM agents. This involves fewer operations, although, the significance of this advantage decreases as the number of agents used increases.

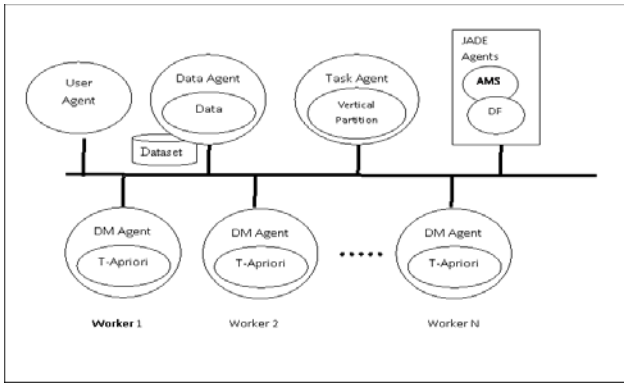


Fig. 4: Parallel/Distributed ARM Model for DATA-VP Task Architecture

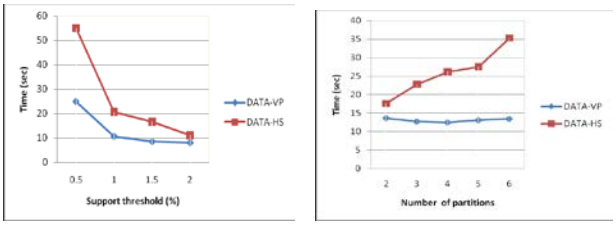
X. EXPERIMENTATION AND ANALYSIS

To evaluate the two approaches, in the context of the MADM vision, a number of experiments were conducted. These are described and analyzed in this section.

The experiments presented here used up to six data partitions and two artificial datasets:

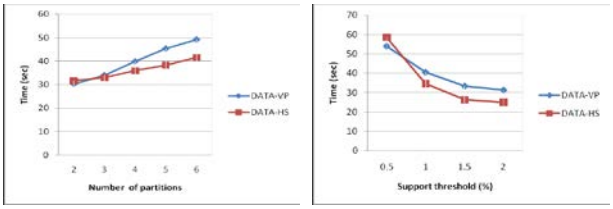
(i) T20.D100K.N250.num, and (ii) T20.D500K.N500.num where T = 20 (average number of items per transactions), D = 100K or D = 500K (Number of transactions), and N = 500 or N = 250 (Number of items) are used. The datasets were generated using the IBM Quest generator used in Agrawal and Srikant [1].

As noted above the most significant overhead of any parallel/distributed system is the number and size of messages sent and received between agents. For the DATA-VP approach, the number of messages sent is independent of the number of levels in the T-tree; communication takes place only at the end of the tree construction. DATA-VP passes entire pruned sub (local) T-tree branches. Therefore, DATA-VP has a clear advantage in terms of the number of messages sent.



(a) Number of Data Partitions (b) Support Threshold

Fig. 5: Average of Execution Time for Dataset T20.D100K.N250.num



(a) Number of Data Partitions (b) Support Threshold

Fig. 6: Average of Execution Time for Dataset T20.D500K.N500.num

Fig. 5 and Fig. 6 show the effect of increasing the number of data partitions with respect to a range of support thresholds. As shown in Fig. 5 the DATA-VP algorithm shows better performance compared to the DATA-HS algorithm. This is largely due to the smaller size of the dataset and the T-tree data structure which: (i) facilitates vertical distribution of the input dataset, and (ii) readily lends itself to parallelization/distribution. However, when the data size is increased as in the second experiment, and further DM (worker) agents are added (increasing the number of data partitions), the results shown in Fig. 6, show that the increasing overhead of messaging size outweighs any gain from using additional agents, so that parallelization/distribution becomes counter productive. Therefore DATA-HS showed better performance from the addition of further data agents compared to the DATA-VP approach.

XI. DISCUSSION

MADM can be viewed as an effective distributed and parallel environment where the constituent agents function autonomously and (occasionally) exchange information with each other. The MADM system is designed with

asynchronous, distributed communication protocols that enable the participating agents to operate independently and collaborate with other peer agents as necessary, thus eliminating centralized control and synchronization barriers.

Distributed and parallel DM can improve both efficiency and scalability first by executing the DM processes in parallel improving the run-time efficiency and second, by applying the DM processes on smaller subsets of data that are properly partitioned and distributed to fit in main memory (a data reduction technique).

The scenario, described in this paper, demonstrated that MADM provides suitable mechanisms for exploiting the benefits of parallel computing; particularly parallel data processing. The scenario also demonstrated that MADM is suitable for re-usability and illustrated how it is supported by re-employing the meta ARM task agent, described in the previous paper, with the DATA-HS task.

XII. Conclusion

In this paper a MADM method for parallel/distributed ARM has been described so as to explore the MADM issues of scalability and re-usability. Scalability is explored by parallel processing of the data and re-usability is explored by reemploying the meta ARM task agent with the DATA-HS task.

The solution to the scenario considered in this paper made use of a vertical data partitioning or a horizontal data segmentation technique to distribute the input data amongst a number of agents. In the horizontal data segmentation (DATA-HS) method, the dataset was simply divided into segments each comprising an equal number of records. Each segment was then assigned to a data agent that allowed for using the meta ARM task when employed on a MADM system. Each DM agent then used its local data agent to generate a complete local T-tree for its allocated segment. Finally, the local T-trees were collated into a single tree which contained the overall frequent itemsets.

The proposed vertical partitioning (DATA-VP) was facilitated by the T-tree data structure, and an associated mining algorithm (Apriori-T), that allowed for computationally effective parallel/distributed ARM when employed on the MADM system.

The reported experimental results showed that the data partitioning methods described are extremely effective in limiting the maximal memory requirements of the algorithm, while their execution time scale only slowly and linearly with increasing data dimensions. Their overall performance, both in execution time and especially in memory requirements has brought significant improvement.

References

[1] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The Quest Data Mining System. In Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining, (KDD), 1996.

[2] R. Agrawal and J. Shafer. Parallel mining of association rules. In Proceedings of the IEEE Transactions on Knowledge and Data Engineering 8(6), pages (962-969), 1996.

[3] K. A. Albashiri, F. Coenen, and P. Leng. EMADS: An Extendible Multi-Agent Data Miner, volume XXIII, pages (263-276). Research and Development in Intelligent Systems, AI, Springer, London, England, 2008.

[4] D. Cheung and Y. Xiao. Effect of Data Distribution in Parallel Mining of Associations. In Proceedings of the Data Mining and Knowledge Discovery 3(3), pages (291-314), 1999.

[5] F. Coenen and P. Leng. Optimising Association Rule Algorithms Using Itemset Ordering. In Proceedings of the AI Conference, Research and Development in Intelligent Systems XVIII, Springer, pages (53-66), 2001.

[6] F. Coenen, P. Leng, and S. Ahmed. T-Trees, Vertical Partitioning, and Distributed Association Rule Mining. In Proceedings of the IEEE International Conference on Data Mining (ICDM), Florida, eds. X Wu, A Tuzhilin and J Shavlik: IEEE Press, pages (513-516), 2003.

[7] F. Coenen, P. Leng, and G. Goulbourne. Tree Structures for Mining Association Rules, volume 8(1), pages (25-51). In the Journal of Data Mining and Knowledge Discovery, 2004.

[8] E. Han, G. Karypis, and V. Kumar. Scalable Parallel Data Mining for Association Rules. In Proceedings of the ACM(Association for Computer Machinery)- Special Interest Group on Management of Data (SIGMOD), International Conference on Management of Data, ACM Press, pages (277-288), 1997.

[9] S. McConnell and D. Skillicorn. Building predictors from vertically distributed data. In Proceedings of the 2004 conference of the Centre for Advanced Studies conference on Collaborative research 04-07. Markham, Ontario, Canada, pages (150-162), 2004.

[10] D. Michie, D. Spiegel halter, and C. Taylor. Machine Learning, Neural and Statical Classification. In Ellis Horwood Series in Artificial Intelligence, New York, 1994.

[11] S. Parthasarathy, M. Zaki, and W. Li. Memory Placement Techniques for Parallel Association Mining. In Proceedings of the 4th International Conference on Knowledge Discovery in Databases (KDD), AAAI Press, pages (304-308), 1998.

[12] T. Shintani and M. Kitsuregawa. Hash Based Parallel Algorithms for Mining Association Rules. In Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, (PIDS), IEEE Computer Society Press, pages (19-30), 1996.

[13] M. Tamura and M. Kitsuregawa. Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster Systems. In Proceedings of the 25th Very Large Data Bases (VLDB) Conference, Morgan Kaufman, pages (162-173), 1999.

[14] A. Poggi, F. Bellifemine and G. Rimassi. JADE: A FIPA-Compliant agent framework. In Proceedings the Practical Applications of Intelligent Agents and Multi-Agents, pages (97-108), 1999. http://www.jade.tilab.com.

[15] G. Piatetsky-Shapiro, U. Fayyad, P. Smyth, and R. Uthurusamy. Advances in Knowledge Discovery and Data Mining. the Association for the Advancement of Artificial Intelligence (AAAI) Press/MIT Press, 1996.

[16] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 Algorithms in Data Mining, Knowledge and Information Systems, volume 14, pages (1-37). Springer-Verlag, London Limited, 2008.

[17] S. Volman, P. Skobelev, I. Minakov, and G. Rzevski. Dynamic Pattern Discovery using Multi-Agent Technology Proceedings of the 6th WSEAS Int. Conference on TELECOMMUNICATIONS and INFORMATICS, Dallas, Texas, USA, March 22-24, 2007.

[18] H. Xargupta, I. Hamzaoglu, and B. Stafford. Scalable, Distributed Data Mining Using an Agent Based Architecture. Proceedings of Knowledge Discovery and Data Mining, AAAI Press, pages (211-214), 1997.

[19] J. A. Yota, A. F. Gmez-Skarmeta, M. Valds, and A. Padilla. Metala. A meta-learning architecture. Fuzzy Days, pages (688-698), 2001.

[20] A. Zurinsky and R. Grossman. A framework for finding distributed data mining strategies that are intermediate between centralized strategies and in-place strategies. In KDD Workshop on Distributed Data Mining, 2000.

[21] Ning Zhou, Kun Gao, Meiqun Liu, Kexiong Chen, and Jiaxun Chen. Sampling-Based Tasks Scheduling in Dynamic Grid Environment.

Proceedings of The 5th WSEAS Int. Conf. On Simulation, Modeling And Optimization, Corfu, Greece, August 17-19, 2005 (Pp25-30).

[22] M. Pejic Zach, N. Vlahovic, B. Knezevic. Public Data Retrieval with Software Agents for Business Intelligence. Proceedings of the 5th WSEAS Int. Conf. on Applied Informatics and Communications, Malta, September 15-17, 2005 (pp215-220).

[23] Y. Chang Zu, T. Yi Lu, R. Fang. An Adaptive E-Learning System Based on Intelligent Agents. Proceedings of the 6th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 15-17, 2007.