Parallel Data Mining Algorithms for Distributed Machines

Aiyesha Ma

CSE 664 – Parallel and Distributed Processing

Abstract

This paper discusses various data mining algorithms and their application to parallel and distributed machines. An overview of common data mining algorithms is given first, then parallelization methods are discussed. An example of a parallelized algorithm is shown for association rule mining. This paper ends with a review of several papers pertaining to parallel pattern discovery. Although some papers opt for a shared memory approach, many have developed and analyzed algorithms for distributed memory machines. The distributed message passing schemes are more suitable for the current trend towards grid computing.

1 Introduction

Cheaper hardware has resulted in an explosion of data collection. Oftentimes, once collected, this data is left to sit because manual analysis is infeasible or impractical. Data mining is the methodological process which seeks to understand massive amounts of data. The goal of data mining is to extract novel and useful information or hidden patterns.

Parallel and distributed processing is important for large scale data mining because of two factors. First, the computational requirements are often very large, and parallel and distributed processing would help in terms of the speed. Second, the data is often stored across multiple storage devices for various reasons such as security or size. [1]

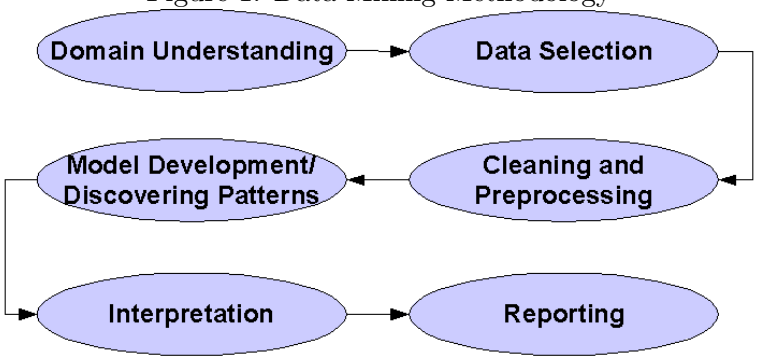
This paper will look at various parallel data mining algorithms currently in the literature. The focus will be on identifying scalable algorithms suitable for distributed environments, in particular those that may be adapted to the grid environment and infrastructure.

The paper is organized as follows. Section 2 gives an overview of data mining, and presents a discussion of the more common data mining algorithms. Section 3 discusses parallelization of data mining algorithms. A review of papers pertaining to data mining on parallel or distributed machines is given in 4. Finally, a summary and concluding remarks are given in 5.

2 Overview of Data Mining

Data mining is a six stage, interactive process (Figure 1). The first step consists of domain understanding. This includes understanding the terminology and problems, and often involves an interactive discussion with domain experts. The second step is to select the relevant data from the data base. A database may include data that is not relevant to the problem to be solved and domain understanding is necessary to choosing the relevant data

Figure 1: Data Mining Methodology



subset. This step will also include data understanding, which often consists of collecting basic statistical properties to better understand the data. This analysis of the data is useful in the next step when performing data cleaning. The third step consists of cleaning and preprocessing. This may include removing outliers and normalization or discretization of the values. The fourth consists of the pattern discovery stage. The remainder of this paper will focus on the different techniques for this stage. The fifth step pertains to interpreting the model results. This stage also consists of verifying the results with a domain expert to make sure the results make sense. The final stage is the dissemination of the knowledge gained.

The pattern discovery stage consists of several approaches, any of which may be utilized to discover information. These approaches consist of classification, clustering, regression, association discovery, sequence analysis, and visualization. Each of these is briefly described below.

Classification Classification is used to put records into predetermined categories. For example, a customer applying for a loan could be classified as either a good credit customer, someone who will pay back the loan on time, or a bad credit customer, someone who will default on the loan. Classification can also be used to separate the historical data in order to gain insight as to which characteristics are important for determining class. For example, older patients undergoing cardiac surgery are slightly more likely to have post-operative complications. Popular classification techniques include Support Vector Machines (SVM), Neural Networks, Decision Trees, and Bayesian decisions.

Regression Regression is similar to classification in that it seeks to determine an output given a set of input variables. Regression, in contrast to classification, determines the output along a continuous scale, rather than in discrete categories. Similar to classification, the resulting prediction model can be analyzed to provide insight to the data. For example, production line information pertaining to silver in the glass coating process is important in determining the end quality of the glass. Commonly used methods also include Neural Networks and Decision Trees along with regression methods such as multiple linear regression and logistic regression.

Clustering Clustering seeks to partition the data into similar groups. Unlike classification, these groups are not predetermined. Furthermore, the number of groups is not always known, further complicating the issue. Clustering is also known as exploratory data analysis. Sometimes the term "unsupervised learning" is used to refer to the fact

the records are not labelled. Popular methods lie in the areas of statistical pattern recognition and neural networks.

Association Discovery In association discovery the goal is to identify relationships between items in database records. An example of an associative relationship might be "90% of customers buying A also buy B". This information is useful in targeted marketing. For example, suppose an analysis of supermarket transactions resulted in the information that many people who buy chips also buy salsa. The salsa could then be conveniently placed along side the chips either in hopes of making the customers happier that they don't have to walk across the store, or in hopes of generating more sales of salsa. Various statistical, machine learning, and neural network approaches have been developed for finding these associations.

Sequence Analysis Sequence analysis is similar to association discovery, except that the associations of interest are ordered and have a time component. An example might be "90% of customers who buy A are likely to buy B in the next year". This information may be used to better target the likely buyers. Another usage example is identifying stolen credit cards. By predicting likely credit card purchases, credit card usage far outside the norm may be expected to be fraud related. Again methods in statistics and neural networks have been developed to find these associations.

Visualization Visualization seeks to exploit the human ability to perceive patterns in visual information. By presenting the data in various forms, the human cognition system is able to extract useful information or patterns from the data.

3 Parallel Data Mining

Parallel data mining algorithms can be distinguished by several characteristics, these include the type of parallelism, task versus data, and the type of load balancing, static versus dynamic. [2] also make the distinction based on the type of memory, distributed versus shared. Although both shared memory and distributed memory designs will be mentioned, the focus of this paper will be on distributed memory designs rather than shared memory algorithms, since most movement is towards cluster and grid computing which is a distributed architecture.

Distributed and Shared Memory Architectures

Distributed Memory The goal when designing algorithms on a distributed memory architecture is to minimize the communication. Communication costs can substantially reduce the performance speedup obtained from parallelization.

Shared Memory The goal when designing algorithms on a shared memory architecture is to minimize synchronization. Good data locality is essential for this. Again large overhead costs can reduce the effectiveness of the parallelism.

Static and Dynamic Load Balancing

Static Load Balancing Generally static load balancing assumes a dedicated homogeneous environment. Typically load balancing is done by dividing data or tasks evenly across all processing nodes.

Dynamic Load Balancing In dynamic load balancing, the algorithm or infrastructure will redistribute tasks from heavily loaded processing nodes to lightly loaded nodes. Dynamic is necessary in a grid environment where processing nodes are heterogeneous and not dedicated.

Data and Task Parallelism

Data Parallel In data parallel algorithms the data is logically or physically distributed across processors. Each processor then performs the same computation on disjoint data sets.

Task Parallel Task parallel algorithms provide all processors with access to the entire dataset. Each processor performs a different task or computation on all data elements.

3.1 Example of Data versus Task Parallelism

A common clustering technique is called K-means clustering. It it the data is clustered into K groups. This is accomplished by an iterative process. The centers are first initialized. Then the data points are compared to the centers to find the most similar and added to the group. After all points have been added to the groups, the center of the groups are recalculated by taking the mean of the groups. The points are iteratively added to the groups, and the centers recalculated until no points change the group they are in, or a maximum iteration is reached. The K-Mean process is outlined in 2.

Figure 2: K-means process

1. Initialize k centers.
2. For each iteration
   (a) For each data point
      i. Compare to center of each group
      ii. Add to group with most similar center
   (b) Compute mean of each group
3. Repeat until no changes to groups, or maximum iteration is reached

To parallelize the algorithm using the data parallel approach, each processor is assigned a disjoint portion of the data points. For example, processor P_1 would be assigned points$\{X_1, ..., X_i\}$, processor P_2 would be assigned points $\{X_{i+1}, ..., X_{2i}\}$, and so on. Each processor would compare its set of data points to all centers. To recompute the groups centers each processor would compute the partial means and either send to all other processors or to a central processor. If sent to a central processor, the central processor would compute the combined means and send back to the other processors. If sent to all processors, each processor individually computes the combined means. Each processor keeps track of changes to the groups (or means) and when there are no more changes to the groups centers across all processors, then the iteration ends. This data parallel approach is shown in 3.

To parallelize the algorithm using the task parallel approach, each processor is given access to all the data points, but each processor only has one center. For example, processor P_1 would have the center of group 1, μ_1, processor P_2 would have the center of group 2, μ_2, and so on. Each processor would then compare all data points to the center it has. Some tracking mechanism would determine to which group each data point was assigned. Each processor would then compute the mean of its group to find the new center. The iterative process would stop when the tracking mechanism determine that there have been no new changes to any groups. This task parallel approach is shown in 4.

Obviously, there are tradeoffs between the two approaches. For example, the task parallel approach would work better in a shared memory environment, than a distributed environment. In a distributed environment the necessary message or data passing would likely have too high an overhead to make the approach beneficial.

Figure 3: K-means process, Data parallel

1. Initialize k centers, distribute to all processors
2. Distribute data to processors
3. For each iteration

P_1	P_2	...	P_p
		...	
(a) For each data point, $x \in \{X_1, ..., X_i\}$	(a) For each data point, $x \in \{X_{i+1}, ..., X_{2i}\}$		(a) For each data point, $x \in \{X_{n-i+1}, ..., X_n\}$
i. Compare to center of each group	i. Compare to center of each group		i. Compare to center of each group
ii. Add to group with most similar center	ii. Add to group with most similar center		ii. Add to group with most similar center
(b) Compute (partial) mean of each group	(b) Compute (partial) mean of each group		(b) Compute (partial) mean of each group
(c) Send partial mean to $P_2, ..., P_p$	(c) Send partial mean to $P_1, P_3, ..., P_p$		(c) Send partial mean to $P_1, ..., P_{p-1}$
(d) Compute actual group centers (total means)	(d) Compute actual group centers (total means)		(d) Compute actual group centers (total means)

4. Repeat until no changes to groups, or maximum iteration is reached

Figure 4: K-means process, Task parallel

1. Initialize k centers, distribute one center to each processor
2. Distribute all data to processors
3. For each iteration

P_1	P_2	...	P_k
(a) For each data point, $x \in \{X_1, ..., X_n\}$ i. Compare to center ii. Mark distance (b) Send all distances to $P_2, ..., P_p$ (c) Select all x in processor group, G_1 (d) Compute group center (mean)	(a) For each data point, $x \in \{X_1, ..., X_n\}$ i. Compare to center of each group ii. Add to group with most similar center (b) Send all distances to $P_1, P_3, ..., P_p$ (c) Select all x in processor group, G_2 (d) Compute group center (mean)	...	(a) For each data point, $x \in \{X_1, ..., X_n\}$ i. Compare to center of each group ii. Add to group with most similar center (b) Send all distances to $P_1, ..., P_{p-1}$ (c) Select all x in processor group, G_k (d) Compute group center (mean)

4. Repeat until no changes to groups, or maximum iteration is reached

3.2 Example of Association Rule Mining

Is association rule mining there are two parts. The first is to find all frequent sets, and the second is to generate the actual association rules. When finding the frequent sets an exhaustive search is infeasible, since, given m items, there are 2^m subsets that might be frequent.

In the search for frequent sets, all sets can be organized on a tree structure in which the bottom nodes are singleton sets containing only one item. Each level, l, in the tree contains sets of size l. When searching, various branches of the tree can be eliminated from the search space based on the fact that given an infrequent set, I, any set, S, containing I is also infrequent. This is the basis for commonly used Apriori algorithm for finding frequent sets. An example of this method follows.

3.2.1 Example

The example given here falls in the area of Market Basket Analysis. In this example six customers have made purchases from a grocery store. The items available in the grocery store are shown in (Table 1). A list of the customer transactions is shown in Table 2.

Table 1: Item Set

Item	Code
Tortilla Chips	T
Salsa	S
Potato Chips	P
Milk	M
Coca-Cola	C

Table 2: Transaction List

Transaction	Items
1	TSM
2	PM
3	TSPM
4	TSC
5	PC
6	TSPC

From the transaction history, the frequent item-sets can be found, and those with a minimum support of 50% are shown in Table 3. The support of an item-set refers to the percentage of occurrences in the transaction history. For example, the item-set TS occurs four out of six times, so it has a support of 67%.

After finding the frequent sets, the association rules can be determined. Association rules with a minimum confidence of 60% are shown for this example in Tabletab:AssocRules. The confidence states the likelihood that a customer will buy a set of items if they are also buying

Table 3: Frequent Item Sets

Support	Item-set
100% (6)	
83% (5)	
67% (4)	T, S, P, TS
50% (3)	C, M

some other set of items. For example, T → S has confidence of 100%. This means that every time a customer buys T, they also buy S.

Table 4: Association Rules

T → S (100%)	C → P (67%)
S → T (100%)	C → S (67%)
	M → P (67%)
	M → S (67%)
	M → T (67%)
	M → TS (67%)

3.2.2 Apriori Algorithm

The Apriori algorithm is an iterative process in which candidate sets are generated, and the support for these sets are found. Succeeding generations then eliminate all candidates with infrequent subsets. The algorithm is stated below:

1. Set the initial candidates as singleton item sets.
2. Scan all transactions to obtain candidate supports
3. Generate candidates of length k from frequent (k-1) length item-sets by a self join on F(k-1)
4. Prune candidates with infrequent subsets
5. Repeat from step 2 until there are no new candidates

The algorithm is illustrated using the previous example set (Figure 5). The initial candidates consist of singleton sets, and the transaction list is scanned to obtain the support of the item-sets (Figure 6(a)). Pairs of the initial sets are joined to produce the candidate sets for the second scan. The second scan then gets the supports for this second set of candidates (Figure 6(b)). The candidates for the third scan would consist of joins between pairs of sets from scan two. All candidates for the third scan would be eliminated because there is no pair of sets from scan two where both sets remain frequent.

3.2.3 Count Distribution Method

A review of parallel association rule mining is given by Zaki in [2]. Although, the paper will be discussed in depth in Section 4.5, an example using one of the algorithms is shown

Figure 5: Sequential Apriori Algorithm, ARM

Candidates	Frequency
T	4
S	4
P	4
M	3
C	3

(a) Scan 1

Candidates	Frequency
TS	4
TP	2
TM	2
TC	2
SP	2
SM	2
SC	2
PM	2
PC	2
MC	0

(b) Scan 2

here. The "count" method is a data parallel method based on the Apriori algorithm. In the "count" method each processor has all candidates, and gets the local supports of those candidates. The processors then exchange their local supports with all other candidates. Each processor is then able to calculate the total support, and to generate the next set of candidates. The algorithm is outlined below:

1. Each processor has F(k-1) candidates
2. For local database partition, Get local support of candidates
3. Exchange local support with all other processors
4. Compute F(k) candidates
5. Repeat until all frequent sets have been found.

Using the previous example, the "count" method is illustrated in Figures 6 and 7. In this example two processors are used, so each processor has three of the transactions. Processor 1 has transactions 1 through 3, and processor 2 has transactions 4 through 6. Again, each processor starts with the candidate set consisting of all singleton set, and the local support is found for each item set. (Figure 6)

After obtaining the local support, the processors exchange their local support with the other processors. Each processor is then able to get the global support of the item sets. Using the global support each processor computes the next candidate set and eliminates those with infrequent subsets. Since no singleton set is infrequent, all candidates are kept, and the processor proceed with the second scan to get the local supports. (Figure 7)

After completing the second scan, the local supports are exchanged. From the computed global support the processor are able to determine that there are no more candidates sets, and the process ends.

4 Review of Applications

This section reviews various papers pertaining to parallel data discovery techniques. The papers are organized according to the data mining tasks mentioned in 2.

Figure 6: Count Method, Parallel Apriori, Transaction Distribution and Scan 1 Local Support

Processor 1

Transaction	Items
1	TSM
2	PM
3	TSPM

Candidates	Frequency
T	2
S	2
P	2
M	3
C	0

Processor 2

Transaction	Items
4	TSC
5	PC
6	TSPC

Candidates	Frequency
T	2
S	2
P	2
M	0
C	3

Figure 7: Count Method, Parallel Apriori, Global Support and Scan 2 Local Support

Processor 1

Candidates	Frequency
T	4
S	4
P	4
M	3
C	3

Cand.	Freq.	Cand.	Freq.
TS	2	SM	2
TP	1	SC	0
TM	2	PM	2
TC	0	PC	0
SP	1	MC	0

Processor 2

Candidates	Frequency
T	4
S	4
P	4
M	3
C	3

Cand.	Freq.	Cand.	Freq.
TS	2	SM	0
TP	1	SC	2
TM	0	PM	0
TC	2	PC	2
SP	1	MC	0

4.1 General Infrastructure

Knowledge Grid — [3] discusses an infrastructure for developing Knowledge Discovery in Databases (KDD) systems on the Grid. They propose using the existing Globus core rather than redeveloping the basic infrastucture features. They define the grid as an "integrated infrastucture for coordinated resource sharing and problem solving in distributed environments". They then define the knowledge grid as "a reference software architecture for the implementation of PDKD systems on top of grid systems such as Globus and Legion."

The goal of the grid infrastructure is to turn distributed heterogeneous computing re-

sources into a single, virtual, parallel computer. While other knowledge grids have been proposed, these are often domain specific, and rarely use existing computational grid services such as authentication, data access, communication, and security services. Cannataro and Talia propose two levels of knowledge grid services. The first layer consists of the core K-grid layer, and contains services directly implemented on generic grid services. The second layer consists of the high level K-grid. The high-level K-grid provides services to describe, develop, and execute PDKD computations.

4.2 Classification

C4.5 Decision Tree — [4] looks at the common C4.5 decision tree algorithm. The paper addresses four issues. The first is the characterization of the C4.5 algorithm, mainly with respect to memory hierarchy usage. The second is the effect of the order of the input dataset on the memory hierarchy. Both of these issues show a memory hierarchy problem. The third issue addressed is the parallelization of the decision tree induction program for a ccNUMA architecture. The authors used RSIM simulator to test various parameters to see how the method operated under various conditions. While other papers compare speed-up relative to a single processor running the parallel method, the authors suggest that instead, speed-up should be relative to the best sequential method, as this will better show the true performance benefits. Lastly the parallelized decision tree is characterized with respect to the memory hierarchy and this result is compared to the single processor version.

When [4] discusses the tree structure in memory, four layouts are mentioned, and two were found to be acceptable for parallel implementation, per-attribute and linked list. This decision was by considering the goal when using a ccNUMA architecture. In ccNUMA, references to nearby local memory are desired. Good data locality will mean less stall time due to memory fetches. The authors note a performance degredation if the stall time is significant. The authors also mention several other parallel decision tree implentations. Three of these, SUBTE, MWK, and MLC, were developed for shared memory (SMP) machines, and exploit task parallelism. Three others, SLIQ, SPRINT, and ScalParC, were developed for distributed message passing architectures, and opt for data parallelism.

In the proposed parallel algorithm, a data parallel approach is used. Furthermore, the tree is grown breadth-first, like ScalParC, rather the depth-first approach used in the actual C4.5 algorithm. The authors state that their proposed parallel algorithm is most similar to ScalParC with the main difference being that ScalParC is on a message passing architecture while the proposed is on a ccNUMA architecture. This difference means that while ScalParC uses a message passing system to exchange information, the proposed algorithm uses an array in shared memory. For both methods, instances are partitioned into new nodes using a has table. This results in significant message passing in the ScalPar system, while the ccNUMA approach simply stores the hash table in main memory. The authors are able to show a respectable speed-up using the parallel approach.

4.3 Regression

Multivariate Regression — [5] acknowledges the problem of distributed data. A method for computing multivariate regression with minimal data aggregation is discussed. The method was applied to linear discriminant analysis and the authors compare the results with those obtained from traditional aggregated data.

4.4 Clustering

Self Organizing Maps — Data and task parallel approaches to self-organizing maps are discussed in [6]. Self-organizing maps are neural network models that map input data vectors into an organized structure of neurons. Generally the neurons are organized in a planar, grid (either square or hexagonal) structure. As a new input is presented to the network, the best responding neuron along with its neighbors are updated to move closer to the presented input. This training process can be accomplished in two ways: on-line and batch. In the on-line training scheme, the neurons are updated after each input is presented. The results of the on-line training scheme are potentially dependent on the order the inputs were presented. In the batch training process, the updates are accumulated and only applied after all inputs have been presented. The authors also mention a modification to the batch training scheme for sparse data sets.

[6] mentions several other parallel approaches to self-organizing maps. These are grouped into task parallel, and data parallel. In the task parallel approaches the neurons are partioned across the processors while in the data parallel approaches the data is partitioned across the processors. The authors state that the task parallel approach has the benefit that it allows the online updating procedure results in exact agreement with the sequential approach. Obermayer et al, Wu et. al., Ceccarelli et. al., and Buhusi were all mentioned by the author as having proposed task parallel approaches.

Mann and Haykin, Ceccarelli et. al., Myklebust and Solheim, and Ienne et. al. were mentioned by the author as having proposed data parallel approaches. The authors state that data parallel approaches have more potential for scalability. The data parallel approach can have either on-line or batch training. The on-line training requirements must be relaxed for a data parallel approach. This results in a difference between the parallel and sequential approaches. The batch training approach doesn't suffer from this drawback. The authors chose to use the batch training approach, and again, extended it to sparse data sets.

The various parallel approaches are compared, and a graph from the paper is shown in Figure 8. This graph shows performance for the data parallel batch approaches, and the task parallel online approach. The speed-up of the task parallel approach is bounded because of the large communication overhead.

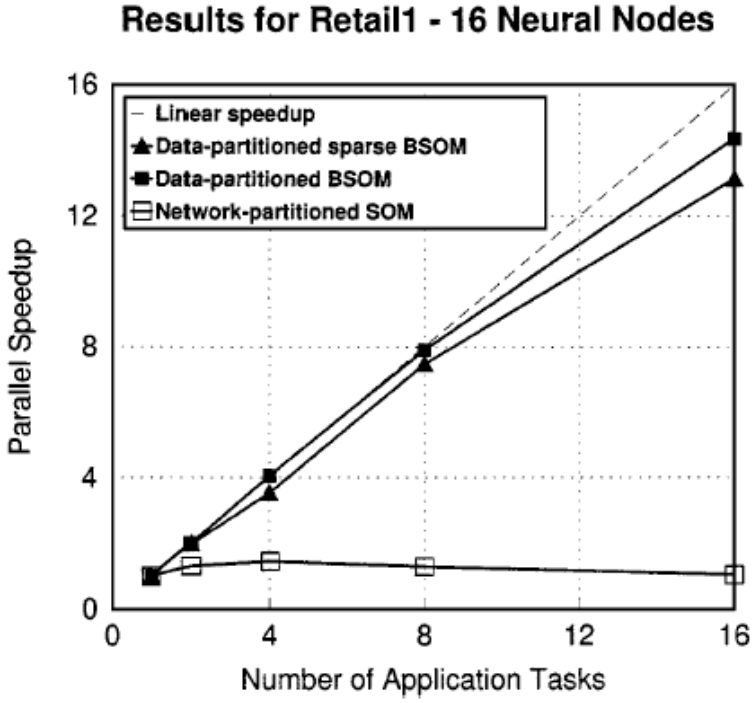
4.5 Association Discovery

Association Rule Mining — A review of association rule mining for parallel processing is given in [2]. The author states having two goals: to present the state-of-the-art, and to identify current open problems. The paper begins with a brief review of the sequential algorithms since the parallel algorithms are based off of these. The sequential algorithms consist of apriori, dynamic hashing and pruning (which is based on apriori), partitioning, other variations on apriori, hashing, and partitioning, dynamic itemset counting (DIC), and various methods proposed by the author.

When discussing the parallel approaches in [2], the author notes that many of the algorithms have been developed for the distributed memory approach rather than the shared memory architectures (SMP). All, or nearly all, the algorithms were based either directly on the apriori algorithm or on one of the apriori variants.

In the shared memory approaches, Zaki proposes two approaches: Common Candidate Partitioned Database (CCPD) and Partitioned Candidate Common Database (PCCD). The second, PCCD, was found too much I/O overhead, and disk contention issues that were

Figure 8: Parallel speed-up with dataset Retail1, on 16 neural nodes, from [6], pg.186



unacceptable. The other, CCPD is a data parallel approach in which an equal distribution of the database is assigned to the processors. Disjoint candidate subsets are generated, and each processor adds information to the common tree, using locks on the leaf nodes. Cheung developed a method called Asynchronous Parallel Mining (APM), which is based on DIC. In APM, transactions are divided into virtual partitions to skew the data via clustering. Skewing the data in this manner allows better pruning of the potential candidates. After partitioning, DIC is applied in parallel on the homogeneous partitions.

In the distributed message passing approaches, Mueller proposes two algorithms, SEAR and SPEAR, based on earlier variations of apriori. Park develops PDM which is based on dynamic hashing and pruning. PDM has a high communication cost because of all to all broadcasts. Agrawal proposed three algorithms based on apriori: count distribution, data distribution, and candidate partitioning. In count distribution there is minimal communication, but the whole hash tree is duplicated in each processor's memory. An example of count distribution was given in Section 3.2. Heuristic partitioning is used in the candidate approach to divide the work equally. This method, however, performs worse that the count distribution approach.

Shintani also proposes several methods similar to Agrawal's: non-partitioned, simply-partitioned, hash-partitioned, and HPA-LD. Non-partitioned is similar to count distribution, simply-partitioned is similar to data distribution, and hash-partitioned is similar to the candidate approach. In HPA-LD extremely frequent itemsets are replicated. Han proposed two approaches: Intelligent Data, and Hybrid. In the intelligent data approach, a linear ring-based all to all communication is used for the first few iterations before switching to the count distribution method. The hybrid approach is a combination between the intelligent data approach and the count distribution method.

Cheung proposed two methods called: Fast Distributed Mining (FDM) and Fast Parallel Mining (FPM). FDM builds on the count distribution method, and in it, the considered candidates are reduced to minimize communication. This is accomplished by several optimizations: local pruning, global pruning, and count polling. In local pruning, itemsets are removed that are not locally frequent. To support global pruning, local supports at k-1 iterations are also sent. Count polling refers to sending local counts to and from home sites. Cheung's other method, FPM, is similar to FDM. In FPM, broadcasts of all local sets are made instead of of just the global sets. This approach removes one message pass per iteration.

Frequent Set Discovery — Generation of frequent item sets is discussed in [7]. They use a lexicographic tree to project transactions. The tree structure begins at level zero with the null set, and expands to one item sets in level one. Expansion continues until all sets are evaluated. Duplicate sets resulting from expansion are eliminated. Each node in the tree is evaluated and inactive nodes, and succeeding branches, are eliminated. The projection method is used in counting supports. If the count is too low, then the set will not have enough support when projected, and is therefore termed inactive. The authors note that the tree structure is smaller than the more common hash structure used in Apriori algorithms.

On uniprocessors, the authors found that the algorithm benefit was minimal for high levels of support, but was much better when looking for itemsets with low levels of support. The authors only mention briefly how the algorithm could be parallelized. They suggest using the "Count Distribution" or "Data Distribution" methods, but use the tree projection approach instead of a hash structure as mentioned in those methods for finding candidate sets.

4.6 Sequence Analysis

Frequent Sequences — [8] presents a parallel algorithm, called pSPADE, for the discovery of frequent sequences. The author discusses the problem of load balancing since the there is no synchronization necessary between processors. Performance is presented with respect to speedup and scalability.

4.7 Visualization

Online Analytical Processing — Online analytical processing (OLAP) is often used for exploratory data analysis, step 2 of the data mining process. [9] discusses a parallel infrastructure for multidimensional OLAP. They also present parallel algorithms for decision tree classification and finding association rules. Performance results are shown for a distributed memory parallel machine with respect to speedup and scalability.

Graph-based Discovery — A graph-based discovery system, SUBDUE, is discussed in [10]. SUBDUE is a graph based discovery approach that captures structural information and presents the information in a way to facillitate the analysis of the data. They use the concept of the minimum description length to compress the data representation. Substructures are found using a computationally constrained beam search. The substructures are then represente by a single node. By finding substructures at various levels, a hierachal description is obtain.

[10] Discusses three methods of parallelizing the SUBDUE approach. Two methods use a data parallel approach while the other uses a task parallel approach. The task parallel approach and one of the data parallel approaches uses dynamic load balancing.

In the task parallel approach each processor is given a substructure or group of substructures to process. A master processor keeps track of the best discovered substructures. In addition the master processor keeps track of a search queue. When a processor has no work, the master processor has busy processor transfer one substructure to the empty processor.

In the dynamic partitioning approach, a data parallel approach, each processor is given a disjoint sets of data. If the processor runs out of work, it requests work from a neighbor processor. The neighbor then passes some work over if it has enough substructures left. A master processor sends the average substructure info regularly to allow individual processors to prune their sets. The master processor also collects and sorts the final list. The authors found there was very little speed-up due to the overhead needed to limit duplicate work, and the overhead for load balancing.

The authors state that the static partitioning approach they propose is the most scalable. In the static partitioning approach, each processor gets a subset of the data, and then performs sequential SUBDUE on the local graph partition. Each processor broadcasts the results from the local graph partition. A master processor collects the results and determines the global best.

5 Conclusion

Developing scalable, parallel data mining algorithms may become necessary in the near future for dealing with the ever increasing amounts of data being archived. However various issues still need to be resolved ([2]). These issues include developing scalable algorithms,

mining of multiple tables, a decentralized approach to information mining, dynamic load balancing, and incremental adjustment methods.

Large databases combined with high dimensionality make scalability an issue. In particular, iterative algorithms that scan an entire database at each iteration may be less scalable than a methodology with fewer scans. Rather than formatting data so that it lies in one table, data mining should be performable on the disparate individual tables. This will be necessary when the dataset is too large to be conveniently reformatted. This also leads into the next issue which is developing decentralized approachs to data mining. The main reason for the decentralized approach is data location. Rather than collecting all the information from various sources into one central location, it may be better to leave the data at the various remote locations for various reasons such as data privacy.

Efficient dynamic load balancing is needed, especially as the movement to grid computing continues. Since computing resources in the grid environment can not be guaranteed, at any given time a computer may become occupied, and its work load will need to be redistributed to other computing nodes. Incremental methods for adding data are also needed, since data never stops being collected.

So, in conclusion, distributed computing could be useful in the analysis of large data collections, but many issues need to be resolved, the most important for many domains probably being security and data privacy.

References

[1] V. Kumar, S. Ranka, and V. Singh, "Guest editor's introduction – special issue on high-performance data mining," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[2] M. J. Zaki, "Parallel and distributed association mining: A survey," *IEEE Concurrency*, vol. 7, no. 4, Oct–Dec 1999.

[3] M. Cannataro and D. Talia, "The knowledge grid," *Communications of the ACM*, vol. 46, no. 1, pp. 89–93, Jan. 2003.

[4] J. P. Bradford and J. e A. B. Fortes, "Characterization and parallelization of decision-tree induction," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[5] D. E. Hershberger and H. Kargupta, "Distributed multivariate regression using wavelet-based collective data mining," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[6] R. D. Lawrence, G. S. Almasi, and H. E. Rushmeier, "A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems," *Data Mining and Knowledge Discovery*, vol. 3, no. 2, pp. 171–195, June 1999.

[7] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, "A tree projection algorithm for generation of frequent item sets," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[8] M. J. Zaki, "Parallel sequence mining on shared-memory machines," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[9] S. Goil and A. Choudhary, "Parsimony: An infrasturcture for parallel multidimensional analysis and data mining," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[10] D. J. Cook, L. B. Holder, G. Galal, and R. Maglothin, "Approaches to parallel graph-based knowledge discovery," *Journal of Parallel and Distributed Computing*, vol. 61, 2001.

[11] V. Breton, R. Medina, and J. Montagnat, "Datagrid, prototype of a biomedical grid," *Methods of Information in Medicine*, vol. 42, no. 2, pp. 143–148, 2003.

[12] C. Kruengkrai and C. Jaruskulchai, "A parallel learning algorithm for text classification," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Press, 2002, pp. 201–206.

[13] V. Kumar and M. Zaki, "High performance data mining (tutorial pm-3)," in *Tutorial notes of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Press, 2000, pp. 309–425.

[14] T. Li, S. Zhu, and M. Ogihara, "A new distributed data mining model based on similarity," in *Proceedings of the 2003 ACM symposium on Applied computing.* ACM Press, 2003, pp. 432–436.

[15] S. Morinaga, K. Yamanishi, and J. Takeuchi, "Distributed cooperative mining for information consortia," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Press, 2003, pp. 619–624.

[16] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Scheduling high performance data mining tasks on a data grid environment," in *Proc. Europar Conference*, 2002.

[17] C. Wang and S. Parthasarathy, "Parallel algorithms for mining frequent structural motifs in scientific data," in *Proceedings of the 18th annual international conference on Supercomputing.* ACM Press, 2004, pp. 31–40.

[18] R. Wright and Z. Yang, "Privacy-preserving bayesian network structure computation on distributed heterogeneous data," in *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Press, 2004, pp. 713–718.

[19] M. J. Zaki and C.-T. H. Ho, "Workshop report: large-scale parallel kdd systems," *SIGKDD Explor. Newsl.*, vol. 1, no. 2, pp. 112–114, 2000.

[20] C. C. Zhang, J. Leigh, T. A. DeFanti, M. Mazzucco, and R. Grossman, "Terascope: Distributed visual data mining of terascale data sets over photonic networks," *Future Generation Computer System*, vol. 19, pp. 935–943, 2003.