Parallelization of Data Mining Algorithms on Computing Grids

Jarret M. Warren\textsuperscript{1} and Wing Ning Li\textsuperscript{2}
\textsuperscript{1}IT Department, Data Tronics, Fort Smith, AR 72903
\textsuperscript{2}Department of computer science and computer engineering, University of Arkansas, Fayetteville, AR 72701

Abstract

Businesses and governments are continuously looking for new and better ways of storing data and then extracting information from these databases.

Data mining is a field of algorithms dedicated to finding useful information in a database, both quickly and accurately. These algorithms tend to be very time consuming, as each record must be examined time and time again. Data mining techniques do exhibit certain qualities that enable them to be parallelized with great performance gains.

A new area of parallel computer design is the computing grid, which networks distributed resources to provide previously unattainable performance and capabilities. Various message-passing protocols, such as MPI, have been developed to provide application programming interfaces. Databases are beginning to be applied to this type of parallel architecture.

This paper proposes a framework for parallelizing various data mining algorithms on computing grids and examines the PAM clustering algorithm in particular and applies it to the computing grid. The parallel implementation outperforms the previous implementation of the algorithm. The improvements made to PAM are then generalized to additional data mining techniques.

1. Introduction

The size of databases has grown rapidly as businesses and governments generate and collect more and more information. Many techniques have been developed to process this data both quickly and intelligently.

Data mining, or knowledge discovery, is a growing field of algorithms for extracting useful information from databases. These algorithms must satisfy certain requirements; among these are the capability to handle different data types, efficiency and scalability, and certainty and expressiveness of the results [5].

Numerous data mining techniques have gained support. Mining association rules, classification, clustering, and Naive Bayes are some examples of knowledge discovery algorithms. A certain technique may be more applicable to a particular type of problem or dataset. Each algorithm balances efficiency with accuracy.

As databases have grown in size, people have begun to look at distributing data over multiple machines. New and creative computer designs have appeared. One of the more popular parallel architectures is the computing grid.

Data mining algorithms are exhaustive in their examination of data. This process can be very time-consuming. Many of these algorithms are open to improvement though. Data mining algorithms typically involve certain core operations, most of which lend themselves to being parallelized. This paper outlines a basic approach for improving these knowledge discovery algorithms.

2. Data Mining Algorithms

A vast number of knowledge discovery algorithms exist. Each approach was created out of the need to solve a certain type of data mining problem.

Mining association rules involve finding connections between attribute-values in a transactional or relational database [2]. A set of strong association rules is created in the form $A_1 \land \ldots \land A_m \Rightarrow B_1 \lor \ldots \lor B_n$, where $A_i$ and $B_j$ are sets of attribute-values. The rules satisfy certain confidence and support requirements. The algorithm begins by finding large itemsets, sets of items meeting a minimum support level, and then using these large itemsets to generate the association rules that achieve a certain support level.

Data classification is a supervised learning technique that classifies a set of data based on the values of certain attributes [11]. The result is a decision tree that categorizes the data. Examples of algorithms performing data classification include ID-3 and C4.5.

Data clustering is an unsupervised learning technique that discovers meaningful partitions of large sets of objects [9]. The goal is to maximize the intraclass similarity and minimize the interclass similarity, utilizing some type of distance function. Example clustering algorithms include k-means, PAM, CLARA, and CLARANS.

Naive Bayes is a supervised learning classification method [4]. This algorithm discovers the probability a record belongs to a given class, provided it contains a certain set of attributes. The Naive Bayes model is:
\( p(x|c) = p(x_1|c) \times \ldots \times p(x_n|c) \) where \( x_i \) is a particular attribute and \( c \) is a particular class.

Data mining often takes the form of a multistage process. The information needed to answer a specific question may be spread across multiple tables in a relational database. Data mining may begin by using queries and joins to produce a single table, or at least a single view, of the required data. Data mining then applies one or more of the knowledge discovery algorithms just described.

### 3. Computing Grids

Computing grids are heterogeneous collections of computers that may reside in different locations, run different software, and be subject to different access control policies [7]. The networking of these distributed computers results in the pooling together of hardware resources. Computing grids lower the cost of computing, extend the availability of resources, increase productivity, and improve quality [10].

Computing grids are relatively new, so grid-enabled tools are still in their infancy. These tools aim to hide the underlying aspects of the grid and to provide an interface for familiar programming models.

One of the more popular tools is MPICH-G, a grid-enabled implementation of the Message Passing Interface (MPI). MPICH-G extends the Argonne MPICH implementation of MPI to use services provided by the Globus grid toolkit [7]. MPICH-G provides a means for managing the resources of the computing grid as well as supplying the basic functions outlined in the MPI standard.

The MPI standard provides a large number of useful functions. MPI is a message-passing application programmer interface [8]. Both point-to-point and collective data transfer operations are available. These instructions include gathering and reducing values among nodes in a grid.

Numerous database management systems, such as Oracle 10g [10], have been created to oversee the distribution and administration of data across computing grids. A database may be distributed in a variety of different ways. The database might be mirrored on several nodes, such that each node has an identical copy of the entire database. Other possibilities include partitioning a single database into individual tables or splitting the database into sections of records or attributes on each node. The choice of data distribution is critical to the performance of queries and knowledge discovery algorithms.

### 4. A Framework for Parallelization

We propose the following framework for parallelizing various data mining algorithms on computing grids. In the framework, data records or databases are partitioned among grid nodes in such a way that static load balancing is achieved. If grid nodes are homogeneous, data records are distributed evenly. To take advantage of in-memory processing over disk-based processing, data records of each grid node reside in the main memory instead of secondary storage such as disk.

Grid nodes are conceptually classified as master(s) and slaves. Master(s) is responsible for coordinating the overall data mining computation; including reducing the partial results obtained by slaves and broadcasting global state information to slaves. Slaves are mainly responsible for carrying out each data mining operations on their portion of the data records. The data parallelism is achieved through slaves working on partitioned data. The major steps of the framework are shown as follows.

1. Data records are partitioned among grid nodes so that static load balancing and main memory residency are met.
2. Master(s) broadcasts current global state to slaves.
3. In parallel, slaves update their local copies of the global state and carry out the needed operations on their portion of the data records.
4. Master initiates the reduction of the partial results obtained by slaves. Repeat steps 2 and 3 until termination criteria are met.

To illustrate the applicability of the proposed framework, one data mining algorithm is used as an example. Based on the framework, we devise a parallel and distributed PAM clustering algorithm and conduct some simulations to test the effectiveness of the approach. A latter section will discuss how other data mining algorithms may be mapped to the model.

### 5. Description of Problem

This paper concentrates on parallelizing the PAM clustering algorithm on a computing grid. Many assumptions have been made. The tables of the database have been preprocessed to produce one table, encompassing all relevant records. These records are distributed across all nodes in the grid in a uniform manner. In addition, the records consist of numerical data points, such that a clearly defined distance metric is available.

### 6. PAM Algorithm

PAM (Partitioning Around Medoids) was developed by Kaufman and Rousseeuw [9]. The PAM algorithm finds clusters of data centered around \( k \) medoids, or representatives. The algorithm identifies the most
centrally located object within each cluster. After the medoids have been found, every object is associated with the nearest medoid.

PAM begins by making a random selection of \( k \) objects to serve as medoids. The algorithm then examines all potential swaps of selected objects, the medoids, with non-selected objects. The current selected object will be referred to as \( O_k \), while the current non-selected object will be referred to as \( O_h \). If an improvement is found, \( O_i \) and \( O_h \) are swapped and the algorithm repeats this entire process. PAM concludes when no improvement can be made to the current clustering.

For each pair of objects that may be swapped, a total cost, \( TC_{ih} \), must be found. This cost is the sum of the cost that each non-selected point would incur if the current set of medoids were to change.

For \( n \) objects and \( k \) medoids, PAM observes \( k \) \((n - k)\) pairs, each requiring the examination of \((n - k)\) objects. For one round of PAM, the complexity is \( O(k(n - k)^2) \). The number of rounds before the algorithm settles is indeterminate.

PAM takes the approach of guessing a solution for the clustering. The algorithm then proceeds to improve upon this solution until no gain can be realized. Depending on the initial set of medoids, PAM may only find a local solution and not the overall best solution. A high-level step by step description of PAM is given below.

1. Select \( k \) representative objects, medoids, arbitrarily and use them to classify the remaining objects and compute the objective function or total cost, which is the sum of dissimilarities between each representative object and the rest objects in the same cluster.
2. For each pair of representative and non-representative objects, \( O_i \) and \( O_h \), compute the total cost change, \( TC_{ih} \), if the roles of the objects were interchanged.
3. If the minimum \( TC_{ih} \) among all pairs obtained in step 2 is negative, interchange the roles of the objects \( O_i \) and \( O_h \). Repeat step 2.
4. Otherwise, use the current representative to classify the non-representative objects. Halt.

7. Parallel PAM Algorithm

The examination of each pair of selected and non-selected objects clearly dominates the runtime of PAM. The proposed parallel implementation of PAM improves on this aspect of the previous algorithm.

The objects in the database are evenly distributed over the computing grid, composed of \( m \) nodes. The goal of PAM is to calculate the cost of each prospective swap, which relies on information from all nodes in the grid. As a result, information must be collected from every node during each round of the algorithm.

The parallel PAM algorithm begins by having each node in the grid randomly select medoids, such that the combined number of medoids across all nodes is \( k \). A global list of medoids is kept at each node. This list, in its entirety, must initially be broadcast to every node.

Each node then proceeds to calculate partial costs for each pair of selected and non-selected objects over all of its local objects. Each current non-selected object must be broadcast to every node when its time comes to be examined. Once all partial costs for swapping \( O_h \) and \( O_h \) are found, partial sums, called \( TC_{ih}[m] \), are computed at each node. When the partial sums for all pairs have been calculated, the nodes perform a reduction of the \( TC_{ih}[m] \)'s to form \( TC_{ih} \). The sums, along with any broadcasts, are supported by MPI’s \texttt{gather} and \texttt{reduce} functions. The MPI standard typically implements a simple tree algorithm to support these types of functions.

If an improvement is found, the pair of objects to be swapped is broadcast to every node. Each node updates its copy of the medoid list. The algorithm continues until no improvement is found.

8. Simulation

Two sequential implementations and one simulated parallel implementation of PAM were created and tested. All code was written in C.

A test dataset was produced using one of Microsoft Excel’s Data Analysis tools. Excel supports simple random number generation, whereby a range of cells can be filled with independent random numbers drawn from one of several distributions, including uniform, normal, binomial, and Bernoulli distributions. This data was then written to a comma delimited file that was later processed by the clustering algorithms.

Two hundred fifty-six objects were created, each representing a two-dimensional point. All points existed on a plane such that the \( x \)- and \( y \)-coordinates ranged from 0 to 100. Four clusters of ten points were created, as well as four clusters of forty points. Each cluster contained points with \( x \)- and \( y \)-coordinates centered on a predetermined point. In addition, fifty-six points were scattered evenly throughout the plane, serving as noise.

For any clustering algorithm, a distance function must exist for determining the similarity between two records, or objects. The database used during testing was composed of two-dimensional points so that the Euclidean distance formula could be applied. Any other distance function could easily be substituted into the algorithm.

The first sequential algorithm calculates and compares
all costs on-the-fly. As each cost is found, it is compared with the previous lowest cost until all pairs of selected and non-selected objects are observed. This version uses minimal storage for costs.

The second sequential algorithm computes the cost for each pair of selected and non-selected objects and stores this value in a data structure. Once this process is complete, the list of costs is examined to find the cost representing the most improvement.

The parallel algorithm, as detailed above, stores partial costs in a data structure at each node and then sums these to obtain a global cost for each pair of selected and non-selected objects. These costs are then examined to find the best swap. The current parallel algorithm implements a binary tree algorithm for the broadcast and the reduction of data, to simulate the best possible case for internode communication. The simulated transmission time between two nodes was designed as a function based on the time an addition instruction takes to execute. This transmission time can be manipulated to create a reasonable approximation of real world performance. The parallel algorithm was tested with one, two, four, eight, or sixteen nodes.

9. Results

Table 1 displays the results of the simulation. Table 2 displays the same results using the speed up measure. The data in the table represents the average of three executions of each algorithm. Each column demonstrates mining for a particular number of clusters. The algorithms were tested with the intent of finding one, two, four, eight, or sixteen clusters. For communication, the parallel simulation utilizes a single transmission time between two nodes that equals the execution time of one thousand addition instructions.

![Table 1](https://example.com/table1.png)

Table 1. Results of Simulation (execution time in seconds)

The results are pretty much as expected. The dominating factor in the sequential algorithm was the calculation of each prospective swap, a step that could be rather easily distributed over a computing grid.

The penalty for storing the costs instead of only keeping track of the best improvement can be seen. The first sequential algorithm outperforms the second sequential algorithm. In addition, the parallel simulation with only one node is the worst performer, as the overhead from the parallel implementation results in some loss of performance.

![Table 2](https://example.com/table2.png)

Table 2. Speed up results using best sequential(on-the-fly)

We should point out to the reader that the results shown are obtained from uncontrolled experimentations, meaning a different instance, which is generated randomly, is used for a different run. Even though the set of data points are the same, the initial medoids are different due to the random selection of medoids. Since the convergence of PAM depends on what the initial medoids are, its run time also depends on the initial medoids. The speed up will be better illustrated if controlled experimentations are used. We hope to perform that in the future.

10. Basic Approach and Additional Application

A well-defined model was developed for parallelizing various data mining algorithms. The approach consists of three primary steps. First, the relevant data is partitioned across a computing grid. Second, the distributed data is processed in a parallel fashion. Lastly, the individual results are collected from all nodes and combined to form a single result.

This approach can be applied to other data mining algorithms when they are used on computing grids. The second step of the model must simply be modified to
compute the values that are necessary for a particular algorithm.

For instance, mining association rules requires counting the number of occurrences of a particular attribute to find large itemsets. These counts can be performed in parallel, and then the counts at each node can be summed up.

Naive Bayes counts the number of occurrences of a particular attribute within a specified class. Once again, these counts of the distributed data could be calculated in parallel during the second stage of the approach, and then combined to form counts across all nodes.

Any data mining algorithm that can be fit to this model should see similar improvements in execution time.

11. Conclusions and Future Work

Various knowledge discovery algorithms were studied. Each algorithm had certain advantages and disadvantages although many shared similar techniques. In addition, the computing grid architecture, as well as the MPI standard, was examined as a platform for parallelizing data mining.

The PAM algorithm was applied to the computing grid environment. A simulation of the parallel implementation of PAM was performed, and the results were compared to two different sequential versions. The parallel implementation clearly outperforms the sequential incarnations. Even with any penalties due to communication and additional storage, the new algorithm is superior.

Time constraints limited the opportunity to apply the parallel PAM implementation to a real-world computing grid. The algorithm would appear to translate rather easily to a version utilizing MPI for a grid. This was the next logical progression in the project.

Other future work includes actual implementations of some additional data mining algorithms, such as association rules and Naive Bayes. Similar improvements can probably be shown. Also, testing the parallel implementations on new, real-world datasets would ultimately demonstrate their capabilities.

References