A Performance Prediction Framework for Grid-Based Data Mining Applications [*]

Leonid Glimcher Gagan Agrawal
Department of Computer Science and Engineering
Ohio State University, Columbus OH 43210
{glimcher,agrawal}@cse.ohio-state.edu

ABSTRACT

For a grid middleware to perform resource allocation, *prediction models* are needed, which can determine how long an application will take for completion on a particular platform or configuration. In this paper, we take the approach that by focusing on the characteristics of the class of applications a middleware is suited for, we can develop simple performance models that can be very accurate in practice.

The particular middleware we consider is FREERIDE-G (FRamework for Rapid Implementation of Datamining Engines in Grid), which supports a high-level interface for developing data mining and scientific data processing applications that involve data stored in remote repositories. The FREERIDE-G system needs detailed performance models for performing resource selection, i.e., choosing computing nodes and replica of the dataset. This paper presents and evaluates such a performance model. By exploiting the fact that the processing structure of data mining and scientific data analysis applications developed on FREERIDE-G involves generalized reductions, we are able to develop an accurate performance prediction model.

We have evaluated our model using implementations of three well-known data mining algorithms and two scientific data analysis applications developed using FREERIDE-G. Results from these five applications show that we are able to accurately predict execution times for applications as we vary the number of storage nodes, number of nodes available for computation, the dataset size, the network bandwidth, and the underlying hardware.

1. INTRODUCTION

A major goal of grid computing is enabling applications to identify and allocate resources dynamically. This can provide end-users flexibility and transparency in executing applications, and ability to solve large problems. However, for a middleware to perform resource allocation, *prediction models* are needed, which can determine how long an application will take for completion on a particular platform or configuration.

In general, such prediction models can be extremely hard to build. In this paper, we take the approach that by focusing on the characteristics of the class of applications a middleware is suited for, we can develop simple performance models that can be very accurate in practice.

The work presented here has been done in the context of FREERIDE-G (FRamework for Rapid Implementation of Datamining Engines in Grid), which supports a high-level interface for developing data mining and scientific data processing applications that involve data stored in remote repositories [12]. The overall motivation for this middleware is as follows. Scientific discoveries are increasingly being facilitated by analysis of very large datasets distributed in wide area environments. Careful coordination of storage, computing, and networking resources is required for efficient dataset analysis. Even if all data is available at a single repository, it is not possible to perform all analysis at the site hosting such a shared repository. Networking and storage limitations make it impossible to down-load all data at a single site before processing.

Thus, an application that processes data from a remote repository needs to be broken into several stages, including a data retrieval task at the data repository, a data movement task, and a data processing task at a computing site. Because of the volume of data that is involved and the amount of processing, it is desirable that both the data repository and computing site may be clusters. This can further complicate the development of such data processing applications. An important goal of the FREERIDE-G system is to enable efficient processing of large scale data mining computations. It supports use of parallel configurations for both hosting the data and processing it. Moreover, in a distributed environment, resources may be discovered dynamically, which implies that a parallel application should be able to execute on a variety of parallel systems. Thus, one of the features of the FREERIDE-G system is to support execution on distributed memory and shared memory systems, as well as on cluster of SMPs, starting from a common high-level interface. FREERIDE-G is also designed to make data movement and caching transparent to application developers.

An important challenge in processing remote data is to allocate computing resources. Additionally, if a dataset is replicated, we also need to choose a replica for data retrieval. Thus, the FREERIDE-G system needs detailed performance models for carrying out such resource selection. This paper presents and evaluates such a performance model. By focusing on the processing structure of the data mining and scientific data analysis applications developed on FREERIDE-G, we are able to develop an accurate performance prediction model. Our model uses *profile* information, i.e., information from the execution of the application on one configuration and dataset size. Using this information, we are able to predict performance on other configurations and dataset sizes. Because FREERIDE-G supports applications that perform generalized reductions only, we are able to accurately model interprocessor communication and the sequential global reduction component.

We have evaluated our model using implementations of three well-known data mining algorithms and two scientific data analysis applications. Results from these five applications show that we are able to accurately predict execution times for applications as we vary the number of storage nodes, number of nodes available for computation, the dataset size, the network bandwidth, and the underlying hardware. Thus, we show that by focusing on a specific class of applications,

[*]This work was supported by grants #CCF-0541058 and CNS #0403342

accurate performance models can be generated.

The rest of this paper is organized as follows. Background information on the FREERIDE-G middleware is provided in Section 2. Our performance prediction framework is described in Section 3. We give an overview of the applications we have used in Section 4, and evaluate our framework in Section 5. Finally, we compare our work with related research efforts in Section 6 and conclude in Section 7.

2. MIDDLEWARE OVERVIEW

This section gives an overview of the design and API of the middleware. More details are available in a recent paper on FREERIDE-G [12] and publications on the predecessor system, FREERIDE [19].

2.1 System Design

The system is designed to automate retrieval of data from remote repositories and coordinate parallel analysis of such data using computing resources available in a grid environment. This system expects data to be stored in chunks, whose size is manageable for the repository nodes.

This middleware is modeled as a *client-server* system. Figure 1 shows the three major components, including the *data server*, the *compute node client*, and a *resource selection framework*. The data server runs on every on-line data repository node in order to automate data delivery to the end-users processing node(s). More specifically, it has 3 roles:

- *Data retrieval:* data chunks are read in from repository disk.
- *Data distribution:* each data chunk is assigned a destination – a specific processing node in the end-user's system.
- *Data communication:* after destination assignment is made in the previous step, each data chunk is sent to the appropriate processing node.

A compute server runs on compute node, with the goal of receiving the data from the on-line repository and performing application specific analysis of it. This component has 4 roles:

- *Data Communication:* data chunks are delivered from a corresponding data server node.
- *Data Retrieval:* if caching was performed on the initial iteration, each subsequent pass retrieves data chunks from local disk, instead of receiving it via network.
- *Computation:* Application specific data processing is performed on each chunk.
- *Data Caching:* if multiple passes over the data chunks will be required, the chunks are saved to a local disk.

The current implementation of the system is configurable to accommodate N data server nodes and M user processing nodes between which the data has to be divided, as long as $M \geq N$. The reason for not considering cases where $M < N$ is that our target applications involve significant amount of computing, and cannot effectively process data that is retrieved from a larger number of nodes. Active Data Repository (ADR) [3, 4] was used to automate the data retrieval parts of both components.

The resource selection framework is being designed with the following goals:

- *Finding Computing Resources:* This module will interface with existing grid resource services, and will use detailed performance modeling to allocate computing resources that can perform the data processing task most efficiently.
- *Choosing Replica:* The data that needs to be retrieved and processed may be replicated across multiple repositories. In such cases, the resource selection framework will choose the repository which will allow data retrieval, data movement, and data processing at the lowest cost.
- *Finding Non-local Caching Resources:* Many data mining and data processing applications involve multiple passes on data. If sufficient storage is not available at the site where computations are performed, data may be cached at a *non-local site*, i.e., at a location from which it can be accessed at a lower cost than the original repository. The resource selection module is also responsible for identifying such non-local caching sites.

In our current implementation, we have not considered non-local caching of data. Thus, the performance prediction framework we are presenting here will be restricted to choosing computing resources and replica.

2.2 Middleware Interface

FREERIDE-G processing API is based on the observation that a number of popular data mining and scientific data processing algorithms share a relatively similar structure. Their common processing structure is essentially that of *generalized reductions*. The popular algorithms where this observation applies include apriori association mining [1], k-means clustering [16], k-nearest neighbor classifier [14] and artificial neural networks [14]. During each *phase* of these algorithms, the computation involves reading the data instances in an arbitrary order, processing each data instance, and updating elements of a *reduction object* using associative and commutative operators.

In a distributed memory setting, such algorithms can be parallelized by dividing the data items among the processors and replicating the reduction object. Each node can process the data items it owns to perform a *local reduction*. After local reduction on all processors, reduction objects are communicated. Finally, *global reduction* is performed. The middleware API for specifying parallel processing of a data mining algorithm is simplified since we only need to support generalized reductions. Users explicitly provide reduction object and the local and global reduction functions as part of the API.

3. PERFORMANCE PREDICTION FRAMEWORK

In this section, we describe the overall resource allocation problem and then discuss our approach for performance prediction.

The resource selection component of the middleware performs the following two tasks: 1) finding computing resources for processing data, and 2) choosing among the multiple replicas, when applicable. Thus, we have the following problem. We are given a dataset, which is replicated at r sites. We have also identified c different computing configurations where the processing can be performed. We assume a standard grid service can identify such potential resources.

Our goal is to choose a replica and computing configuration pair where the data processing can be performed with the minimum cost. The choice of the configuration pair depends on both the characteristics of the environment, as well as the particular application. For

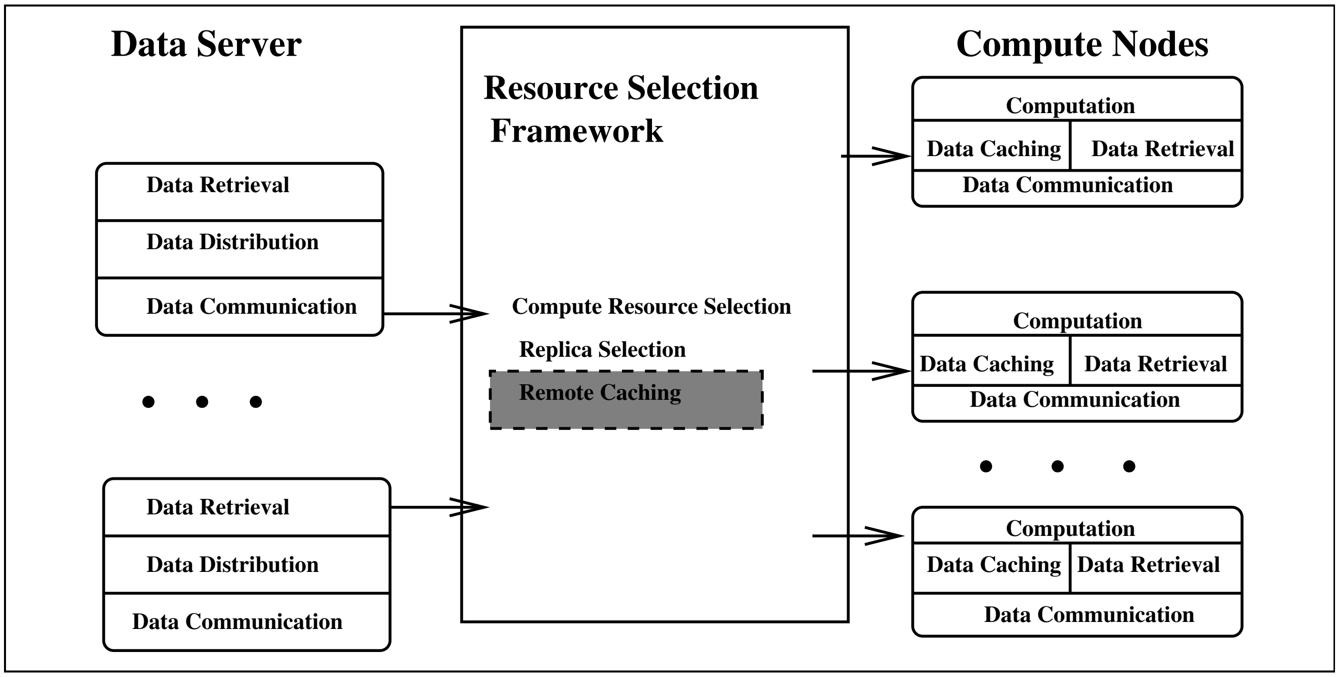

Figure 1: FREERIDE-G System Architecture

example, in one configuration pair, data may be divided across 8 storage nodes, and another 8 nodes may be available for processing. In another configuration pair, data may only be divided into 4 storage nodes, but 16 nodes may be available for processing. For an application where data retrieval cost is a very high, the first configuration pair may be preferable, whereas for a compute-intensive application which can scale well, the second configuration pair may be preferable. Clearly, we can enumerate different possible configurations pairs, and compare the associated costs. Thus, our problem reduces to that of estimating the execution time for a particular configuration.

3.1 High-level Approach

Since remote data analysis applications developed on FREERIDE-G can be split up into data retrieval, data communication and data processing components, our framework models (and predicts) execution time for each component separately. It can combine them all together in order to come up with an approximate execution time for each the resource mapping alternative.

$$T_{exec} = T_{disk} + T_{network} + T_{compute}$$

To fit this model, predictions have to be based on a *profile*, which is collected by executing the application on one dataset and one execution configuration. Based on such execution, we can collect important *summary information*, which can then be used to predict execution times on other configurations and datasets. The summary information comprises of:

- Number of storage nodes (n), compute nodes (c), and the available bandwidth between these (b), in the configuration used for generating the profile.
- A breakdown of the execution time into data retrieval, network communication, and processing components, denoted as t_d, t_n, and t_c, respectively.
- The dataset size, s, which corresponds to the number of elements that are retrieved, communicated, and on which local reduction is performed.
- Maximum size of the reduction object on the dataset used for the profile run.
- Maximum communication time for the reduction object on the configuration used for profile run.
- The global reduction time on the configuration used for the profile run.

When we need to predict across heterogeneous resources, i.e., we use a profile from a cluster with one type of machine and need to predict performance on cluster with another type of machine, we also need scaling factors for CPU, disk, and network across these machines. These scaling factors are computed by executing several representative applications, and are denoted as s_c, s_d, and s_n, respectively.

3.2 Predicting Data Retrieval and Communication Times

Data retrieval time ($\hat{T_{disk}}$) is predicted by scaling the corresponding component (t_d) of the profile execution time breakdown. Since retrieval time primarily depends on dataset size (s) and number of data nodes (n), these pieces of information from the profile configuration (s, n) and from the configuration for which execution time is being predicted ($\hat{s}$, $\hat{n}$) are used to come up with the scaling factors.

$$\hat{T_{disk}} = \frac{\hat{s}}{s} \times \frac{n}{\hat{n}} \times t_d$$

Data retrieval times normally scale very well with the size of the cluster. The expression above assumes that the type of storage nodes does not change between the configurations. If this is not true, i.e., nodes differ in the number of disks or disk speeds, another experimentally determined scaling factor is used, as we will explain in Section 3.4. We are also assuming that retrieval time is linear to the size. We believe it is a reasonable assumption, as long as the size of the dataset is not too small or large.

A similar procedure is used to predict the data communication time ($\hat{T_{network}}$) from the corresponding component (t_n) of the profile execution time breakdown. The only difference is that along with dataset size and number of data nodes, the communication component of the execution time also depends on network bandwidth (b) available between retrieval and compute nodes for communicating data. Therefore, bandwidth information from profile configuration (b) and configuration being predicted ($\hat{b}$) are used to come up with the scale factor.

$$\hat{T_{network}} = \frac{\hat{s}}{s} \times \frac{n}{\hat{n}} \times \frac{b}{\hat{b}} \times t_n$$

The expression above makes several assumptions. First, we are assuming that the throughput increases as the number of storage nodes increases. If this is not the case, the term $\frac{n}{\hat{n}}$ can be removed from the expression above. Second, we are assuming that the bandwidth between storage nodes and computes nodes in any configuration remains constant. This assumption may not often hold in a grid environment. However, in recent years, many efforts have focused on determining the *effective bandwidth* available for a particular data movement task [23, 28, 35, 36]. We can directly use this work to determine $\hat{b}$.

3.3 Predicting Data Processing Time

A similar strategy assuming linear parallel speedups can be used for predicting the data processing time. Data processing time ($\hat{T_{compute}}$) in such situation depends on the number of compute nodes (c) and the size of the dataset. Predicting it, therefore, involves scaling the corresponding component of the profile execution time breakdown (t_c), using numbers of compute nodes from profile configuration (c) and from the configuration being predicted ($\hat{c}$), along the scaling of the dataset size.

$$\hat{T_{compute}} = \frac{\hat{s}}{s} \times \frac{c}{\hat{c}} \times t_c$$

This strategy obviously does not account for inter-processor communication usually involved in parallelizing the processing associated with the applications on FREERIDE-G. Our prediction framework can achieve higher accuracy by modeling interprocessor communication and global reduction times. As described in Section 2, interprocessor communication in FREERIDE-G is restricted to communicating reduction objects after completing local reduction on each node. Global reduction is then performed combining reduction objects from multiple nodes and extracting application specific knowledge from them.

3.3.1 Modeling Interprocessor Communication Time

Time required to communicate the reduction object (T_{ro}) can be modeled as a linear function of the reduction object size (r).

$$T_{ro} = w \times r + l$$

where, w and l are experimentally determined bandwidth and latency for the target processing configuration. Because T_{ro} is a serialized component of the parallel processing time, we can now figure out the fraction of t_c that was can be parallelized:

$$T' = t_c - T_{ro}.$$

The scale factors presented in Section 3.3 can then be applied to T'. After, current overhead ($\hat{T_{ro}}$) is added to make up a new prediction of $\hat{T_{compute}}$.

$$\hat{T_{compute}} = \frac{\hat{s}}{s} \times \frac{c}{\hat{c}} \times T' + \hat{T_{ro}}$$

As one can see, $\hat{T_{compute}}$ depends on $\hat{T_{ro}}$, which, in turn, depends on the size of the reduction object being communicated. As the number of computing processors in the target configuration could be different than the profile configuration, the size of reduction object is unknown. It turns out, however, that size of a reduction object for a particular configuration, can be estimated from the size of the reduction object on the profile configuration, which is provided to the framework.

Our experience with reduction computations shows that almost all applications fall into one of the two classes. For applications in the first class, the reduction object size grows linearly with the number of processing nodes, as well as the dataset size. We refer to this class as *linear object size* class. For applications in the second class, the reduction object size depends only on the application parameters, and does not change with respect to dataset size or the number of processors. We refer to this class as *constant reduction object size*.

Whether an application falls into the linear object size or constant reduction object size class can be determined in one of many ways. A user of the FREERIDE-G can provide this information to the prediction framework. Alternatively, by looking at reduction object size from two or more profile runs with different dataset size and/or processing nodes, we can obtain this information.

The estimated reduction object size can be used to calculate $\hat{T_{ro}}$ and, therefore, $\hat{T_{compute}}$.

3.3.2 Modeling Global Reduction Time

Another component of the data processing time $T_{compute}$ is the time spent in the global reduction phase of computation. Again, this time is serialized, so in order to find the scalable component of the data processing time, the framework subtracts this time from the data processing time component of profile.

$$T'' = t_c - T_{ro} - T_g.$$

Similar to the communication time predictors

$$\hat{T_{compute}} = \frac{\hat{s}}{s} \times \frac{c}{\hat{c}} \times T'' + \hat{T_{ro}} + \hat{T_g}$$

As in the case with T_{ro}, T_g for the profile run is known, but for the configuration being predicted for, it needs to be estimated.

Again, trends observed split applications into two classes. In the first class, T_g scales up linearly with the number of processing nodes, but is independent of the dataset size. In the second class, T_g remains constant as the number of processing nodes is varied, but is linear on the dataset size. This observation allows us to come up with 2 predictors for the global reduction time, the *linear-constant global reduction* time predictor and the *constant-linear global reduction* time predictor. Again, the appropriate predictor for a given application can either be selected by a user, or can be determined by analyzing multiple profile runs.

3.4 Modeling Across Heterogeneous Clusters

So far, the approach described only considers situations where predictions are being made about application behavior on the same set of computing resources (repository and compute clusters) that the base profile information was collected on. However, in practice, clusters can be very different in terms of CPU type and speed, disk speed and configuration, and other factors.

Our approach for predicting performance across different types of clusters is as follows. Suppose we have obtained a profile run for an application on a cluster A. To predict performance on another cluster B, we need a *scaling factor* between nodes in cluster B and cluster A. We believe that a small set of FREERIDE-G applications can be used to experimentally measure the scaling factor between heterogeneous resources. Such a scaling factor can then be used to predict performance for any FREERIDE-G application. Since there is a similarity in the structure of applications that can be supported on FREERIDE-G, we believe that this simple approach can allow reasonably accurate predictions.

As our approach predicts execution times as a sum of T_{disk}, $T_{network}$, and $T_{compute}$, effects of using the new resource configurations are modeled individually for each of these three components. For example, given 2 clusters A and B, we want to predict execution time of an application on cluster B. Suppose we have experimentally obtained execution time breakdowns for three representative FREERIDE-G applications (refered to as 1, 2, and 3) on *identical configurations* on both of these systems. By identical configuration, we mean the same number of compute and storage nodes, and the same dataset size. Now, to compute relative speedup s_d of the data retrieval component of the execution time from A to B , we compute

$$s_d = (\frac{T_{disk_{1-B}}}{T_{disk_{1-A}}} + \frac{T_{disk_{2-B}}}{T_{disk_{2-A}}} + \frac{T_{disk_{3-B}}}{T_{disk_{3-A}}})/3$$

Relative speedups s_n and s_c for data communication and computing stages can also be computed by substituting appropriate components of the execution time into the above equation ($T_{network}$ for s_n and $T_{compute}$ for s_c, respectively).

Given these speedups, we compute predicted performance on the cluster B as follows. Initially, we predict the performance of the application on an identical configuration on the cluster A, including computing the data retrieval time T_{disk_A}, the network time $T_{network_A}$, and the compute time $T_{compute_A}$. Then, we use the scaling factors to predict the overall performance on the new cluster:

$$\hat{T_{exec_B}} = s_d \times \hat{T_{disk_A}} + s_n \times \hat{T_{network_A}} + s_c \times \hat{T_{compute_A}}$$

4. APPLICATIONS

In this section we describe the applications that we have used to carry out the experimental evaluation of our framework. We have focused on three traditional datamining techniques: k-means clustering [15], EM clustering [7], k-nearest neighbor search [14], as well as two scientific feature mining algorithms: vortex analysis [24] and molecular defect detection [29].

4.1 k-means Clustering

The first data mining algorithm we describe is the k-means clustering technique [15], which is one of the most popular and widely studied data mining algorithm. This method considers data instances represented by points in a high-dimensional space. Proximity within this space is used as criterion for classifying the points into clusters. Three steps in the sequential version of this algorithm are as follows: 1) Start with k given centers for clusters, 2) Scan the data instances. For each data instance (point), find the center closest to it, assign this point to a corresponding cluster, and then move the center of the cluster closer to this point, and 3) Repeat this process until the assignment of the points to cluster does not change.

This method can be parallelized as follows. The data instances are partitioned among the nodes. Each node processes the data instances it owns. Instead of moving the center of the cluster immediately after the data instance is assigned to the cluster, the *local sum* of movements of each center due to all points owned on that node is computed. A *global reduction* is performed on these local sums to determine the centers of clusters for the next iteration.

4.2 Expectation Maximization Clustering

The second data mining algorithm we have used is the Expectation Maximization (EM) algorithm [7], which is also a very popular clustering algorithm. EM is a distance-based based algorithm that assumes the data set can be modeled as a linear combination of multivariate normal distributions. The goal of the EM algorithm is to use a sequence of Expectation and Maximization steps to estimate the means C, the covariances R, and the mixture weights W of a Gaussian probability function.

Parallelization of this algorithm on FREERIDE-G [10] is accomplished through iteratively alternating local and global processing, corresponding to each one of E and M steps. During the E step, each node computes the means and the mixture weights of the data instances local to it, followed by this information being gathered at the master node to compute the aggregate, which is then broadcasted. In the M step, covariances of the data instances local to each node are computed, followed by gathering covariance matrices from all processing nodes at the master node, computing a common covariance, and re-broadcasting this information. The algorithm works by successively improving the solution found so far. The algorithm stops when the quality of the current solution becomes stable, which is measured by a monotonically increasing statistical quantity called *loglikelihood*.

4.3 k-Nearest Neighbor Search

k-nearest neighbor classifier is based on learning by analogy [14]. The training samples are described by an n-dimensional numeric space. Given an unknown sample, the k-nearest neighbor classifier searches the pattern space for k training samples that are closest, using the euclidean distance as measure of proximity, to the unknown sample. Again, this technique can be parallelized as follows. The training samples are distributed among the nodes. Given an unknown sample, each node processes the training samples it owns to calculate the k-nearest neighbors *locally*. After this local phase, a global reduction computes the overall k-nearest neighbors from the k-nearest neighbor on each node.

4.4 Vortex Detection Algorithm

Vortex detection is the first of the two scientific data processing applications we have used. Particularly, we have parallelized a feature mining based algorithm developed by Machiraju *et al.*. A more detailed overview of the algorithm is available in a recent publication [34]. The key to the approach is extracting and using *volumetric regions* to represent features in a CFD simulation output. This approach identifies individual points (*detection step*) as belonging to a feature (*classification step*). It then *aggregates* them into regions.

Parallelizing this application requires the following steps [13]. First, a special approach to partitioning data between nodes (overlapping data instances from neighboring partitions) is performed, in order to avoid communication in the detection phase. Detection, classification and aggregation are first performed locally on each node, followed by *global combination* that joins parts of a vortex belonging to different nodes. De-noising and sorting of vortices is performed after the inter-node aggregation has been completed.

4.5 Molecular Defect Detection Algorithm

The second of the two scientific data processing applications we have used performs molecular defect detection [29]. More specifically, its goal is to uncover fundamental defect nucleation and growth processes in Silicon (*Si*) lattices, either in the presence of thermal sources or extra atoms (e.g., additional *Si* atoms or dopants such as Boron). A detection and categorization framework has been developed to address the above need.

This framework is parallelized in the following way [11]. Defect *detection* phase, consisting of marking individual atoms as belonging to defects and clustering them to form defect structures, is paralelized in a manner very similar to vortex detection algorithm. Defects are first detected and aggregated on the chunks of the *Si* grid local to each processing node, followed by joining of defects spanning multiple nodes in the *global combination* stage. Detected defects are then re-broadcasted by the master node, in order to improve load balancing in the categorization phase.

Parallelization of the *categorization* phase, involving computing candidate classes for each detected defect and exact shape matching of the defect to each of the candidate classes is more involved, since it potentially involves a defect catalog update, if no class turns out to be a match. First, all matching defects are categorized locally, and all non-matching defects are given temporary class assignments, which are added to local catalogs. Local catalogs are then merged in the *global combination* step, and after a new copy of the defect catalog is created, its copy is re-broadcasted to compute nodes in order to finalize temporary class assignments.

5. EXPERIMENTAL RESULTS

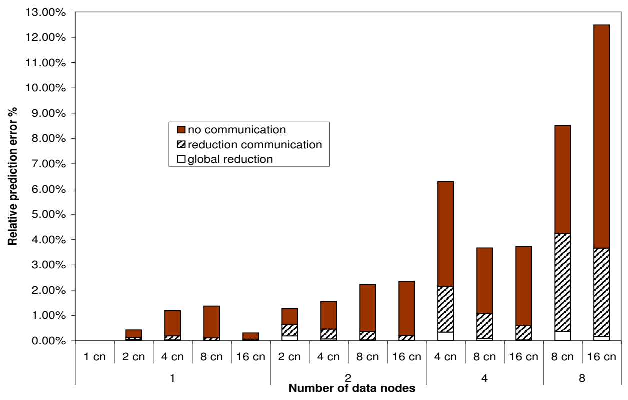

Figure 2: Prediction Errors for k-means Clustering: Base Profile: 1-1, 1.4 GB dataset

In this section we evaluate our execution time prediction techniques for remote data mining applications developed using FREERIDE-G. The techniques were used to predict parallel execution times of five data-intensive applications that were described in the previous section, and were previously implemented using our middleware [12]. Performance prediction models were evaluated in terms of their error relative to the actual execution time.

$$E = \frac{|T_{exact} - T_{predicted}|}{T_{exact}}$$

The goal of our experimental evaluation was to demonstrate that our techniques are well-suited for modeling FREERIDE-G applications in terms of scalability relative to dataset size, parallel scalability relative to both the number of data and compute nodes, changes in data communication bandwidth between data and compute nodes, and change in underlying platform. Initially, we focus on prediction for a cluster with the same kind of machines. Each prediction experiment here is based on an application *profile*, which is a breakdown of execution times for a given configuration of number of data storage/retrieval nodes, number of data processing nodes, dataset size, and the network bandwidth available for data communication. The cluster used for these experiments comprised 700 MHz Pentium machines connected through Myrinet LANai 7.0. Network bandwidth was varied synthetically for some of the experiments. Note that this setup still allowed us to effectively evaluate our models for communication and global reductions, which is the main focus of our effort.

Our last set of experiments focused on evaluating the ability to accurately model predictions on a cluster with different type of machines. For these experiments, the cluster described above was used to obtain base profile information. Predictions were then made for a cluster of dual processor 2.4GHz Opteron 250 machines connected through Mellanox Infiniband (1Gb).

For efficient and distributed processing of datasets available in a remote data repository, we need high bandwidth networks and a certain level of quality of service support. Recent trends are clearly pointing in this direction. However, for our study, we did not have access to a wide-area network that gave high bandwidth and allowed repeatable experiments. Therefore, all our experiments were conducted within a single cluster.

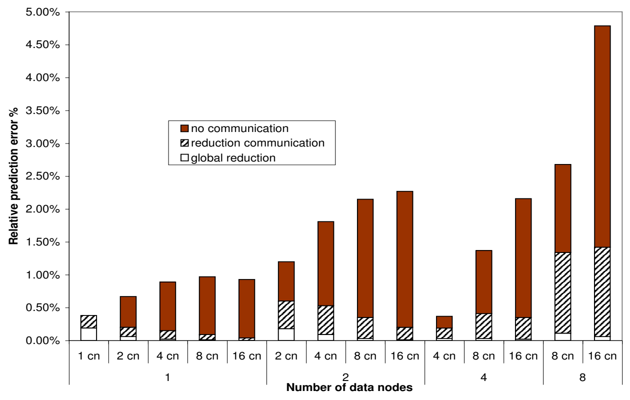

Figure 3: Prediction Errors for Vortex Detection, Base profile: 1-1 710 MB dataset

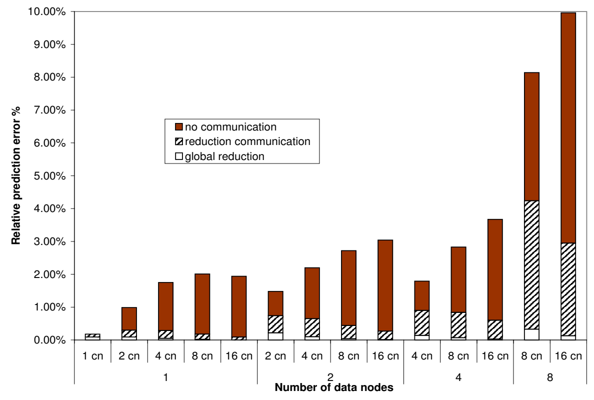

Figure 4: Prediction Errors for Molecular Defect Detection, Base profile: 1-1 130 MB dataset

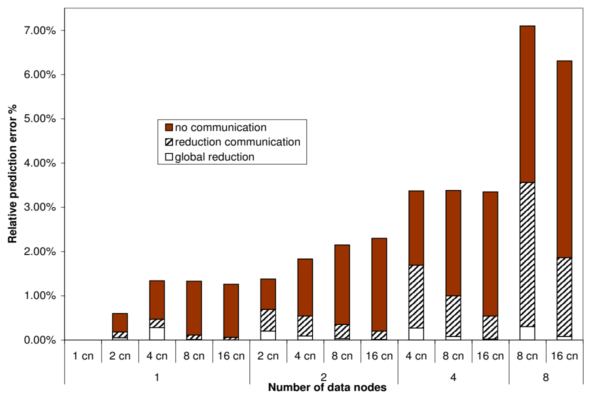

Figure 5: Prediction Errors for EM Clustering, Base profile: 1-1, 1.4 GB dataset

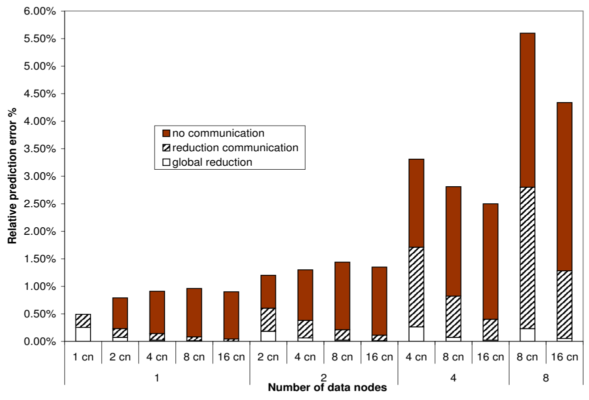

Figure 6: Prediction Errors for KNN search, Base profile: 1-1, 1.4 GB dataset

In all experiments presented here, the number of data nodes is always kept smaller then the number of compute nodes, for reasons mentioned earlier in Section 2. Number of data nodes is varied between 1 and 8, and the number of compute nodes is varied between 1 and 16.

As discussed in Sections 3.3.1 and 3.3.2, for any application, multiple techniques exist for modeling interprocessor communication and global reduction times. In all of the experimental results included in this paper, *linear reduction object size* communication time and *constant-linear global reduction* time prediction approaches are used for vortex detection, molecular defect detection and EM clustering applications. Similarly, *constant reduction object size* communication time and *linear-constant global reduction* time predictor are used for k-means clustering and KNN search applications.

5.1 Modeling Parallel Performance

The first set of experiments we present were designed to show that the prediction framework presented here can be used to correctly model parallel application execution times. We vary only the number of data and compute nodes, and leave the other base profile configuration parameters, i.e. dataset size and network bandwidth, unchanged. Predictions are made for a number of configurations ranging from 1-1 to 8-16, and are all based on a 1-1 configuration profile.

For each configuration, i.e, the number of data nodes and the number of compute nodes, three prediction approaches were compared. The *no communication* approach combines predictors for data retrieval and communication times described in Section 3.2 with the data processing time predictor that does not account for reduction object communication or global reduction, as described at the beginning of Section 3.3. The *reduction communication* approach combines predictors for data retrieval and communication times described in Section 3.2 with the data processing time predictor that models interprocessor communication of the reduction object, as described in Section 3.3.1. Finally, the *global reduction* approach combines predictors for data retrieval and communication times described in Section 3.2 with the data processing time predictor that models interprocessor communication and the global reduction operation, as described in Section 3.3.2.

Figure 2 shows accuracy levels for execution time prediction for k-means clustering. The *no communication* model turned out to be relatively accurate, with predictions from only 3 configurations (4-4, 8-8, and 8-16) resulting in error higher than 4%. The same configurations proved to be the hardest for the *reduction communication* model, with prediction errors for all other configurations being under 2%. The *global reduction* model proved to be extremely accurate for this application, resulting in near-zero errors. Thus, by factoring in communication and global reduction times for data processing, we can accurately predict parallel execution times for this application.

Figure 3 shows the results for vortex detection application. The *no communication* model proved to be quite accurate again, resulting in less than 2% error in all but 4 configurations (2-8, 2-16, 8-8, 8-16). The *reduction communication* model obviously was more accurate, resulting in a prediction error of over 0.5% only for 2 configurations (8-8, 8-16). Again, as we would expect, the *global reduction* model was extremely accurate. Figure 4 presents accuracy of predicting the molecular defect detection execution time. The *no communication* model, again, proved to be quite good, resulting in error higher than 4% in only 2 configurations: 8-8 and 8-16. In case of the *reduction communication* model, 4-4, 8-8, and 8-16 were the only ones where the error exceeded 1%. The *global reduction* model again proved to be very accurate.

Figures 5 and 6 show the accuracy of execution time prediction for EM clustering and KNN search, respectively. The results are very similar to the previous three applications. Overall, this set of results from our five applications show two important trends: 1) even without modeling communication and global reduction, our models work quite well if the scaling factors for number of data storage and computing nodes are small, and 2) our simple models for predicting communication and global reduction times work very well for all cases.

5.2 Modeling Scaling of Dataset Size

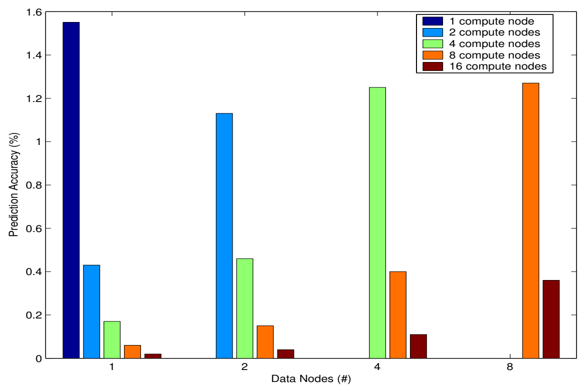

Figure 7: Prediction Errors for EM clustering, 1.4 GB dataset, Base profile: 1-1 with 350 MB

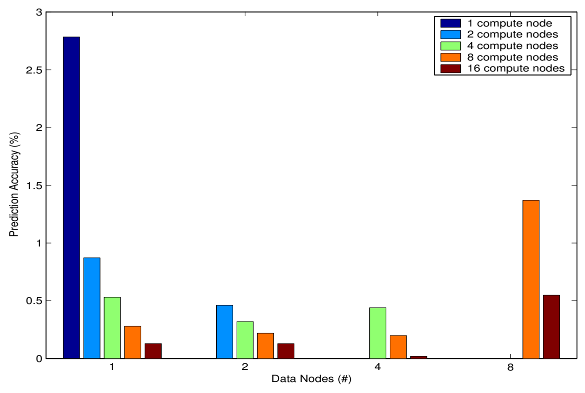

Figure 8: Prediction Errors for Molecular Defect Detection with 1.8 GB dataset, Base profile: 1-1 with 130 MB

The second set of experiments included in this paper were designed to show that the prediction framework presented here can also be used to correctly model the effects of scaling dataset size on application execution time. As in the first set of experiments, the base profile used was the 1-1 configuration, i.e., with 1 data node and 1 compute node. Predictions were made for a number of configurations ranging from 1-1 up to 8-16. The profile runs were with a smaller dataset, and the actual executions were with a larger dataset.

As we noticed in the previous subsection, the *global reduction* technique always yields higher accuracy than the other two. Therefore, in the rest of this section, we present accuracy results using only the global reduction approach. Also, because our prediction approach yields very similar results for all applications that we dealt with, and because of space limitations, we only present results from a subset of applications.

Figure 7 presents accuracy resulting from execution time prediction for the EM clustering application. Dataset size is 350 MB for the base profile configuration and 1.4 GB for the configuration for which execution time is to be predicted. Such change in dataset size leaves the shape of the relative error plot unchanged, i.e. relative to the Figure 5, where the dataset size was unchanged.

Although the actual error has increased somewhat, our models still give very close approximations of execution times. Somewhat higher errors (but still under 2%) are observed for configurations where the numbers of data and compute nodes are equal, but these errors actually drop off for configurations where the number of compute nodes is scaled up. The reason for this is that overestimation performed by our compute time prediction model that shows up for configurations with equal numbers of data and compute nodes is offset by our underestimation in modeling compute node scale-up.

Figure 8 presents accuracy resulting from execution time prediction for the molecular defect detection application, using the *global reduction* approach. Dataset size is 130 MB for the base profile configuration and 1.8 GB for the configuration for which execution time is to be predicted. Once again, although the relative shape of the plot remains unaffected by differences in dataset size, highest prediction errors are still observed for configurations where the numbers of data and compute nodes are equal. Again, a drop off is observed for configurations where the number of compute nodes is scaled up. Among configurations with equal numbers of compute and data nodes, the ones with 2 and 4 compute nodes demonstrate considerably smaller errors than one with 8 compute nodes. This is because this particular application scales linearly when number of data nodes is 2 or 4, but only demonstrates a sub-linear speedup once the number of data nodes is increased beyond that.

5.3 Impact of Network Bandwidth

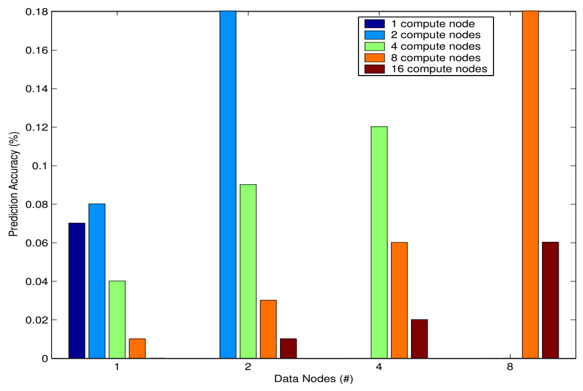

Figure 9: Prediction Errors for Molecular Defect Detection with 250 Kbps, Base profile: 1-1 with 500 Kbps

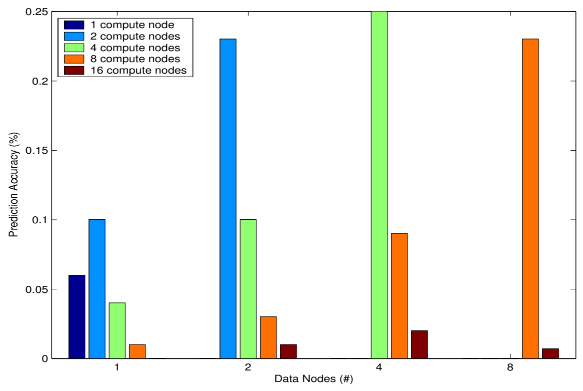

Figure 10: Prediction Errors for EM Clustering with 250 Kbps, Base profile: 1-1, with 500 Kbps

The purpose of the third set of experiments is to show that our approach can successfully model the impact of changing data communication bandwidth. Like the first 2 sets, the base profile used was the 1-1 configuration and predictions were made for a range of configurations. The predictions do not have the same data communication bandwidth, as the corresponding base profile configurations, but the dataset size is the same in both. Again, results of only the *global reduction* approach are presented for the same subset of applications as in the previous subsection.

Results of this set of experiments are summarized in Figure 9 for the defect detection application and Figure 10 for the EM clustering application. Again, the least accurate predictions correspond to configurations where the numbers of data and compute nodes are equal. The shape of the accuracy graph suggests that scaling the number of data nodes doesn't necessarily result in a perfectly linear speedup, as modeled by our approach. However, as the number of compute nodes is scaled up, the effects of inaccuracies in our model are offset by errors in modeling compute node scale-up.

5.4 Predictions for a Different Type of Cluster

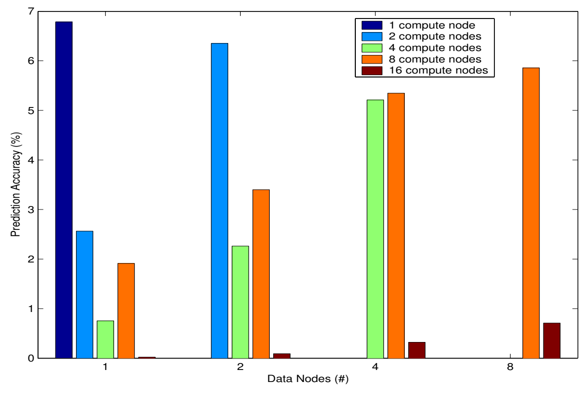

Figure 11: Prediction Errors for EM clustering On a Different Cluster, 700 MB dataset, Base profile: 8-8 with 350 MB

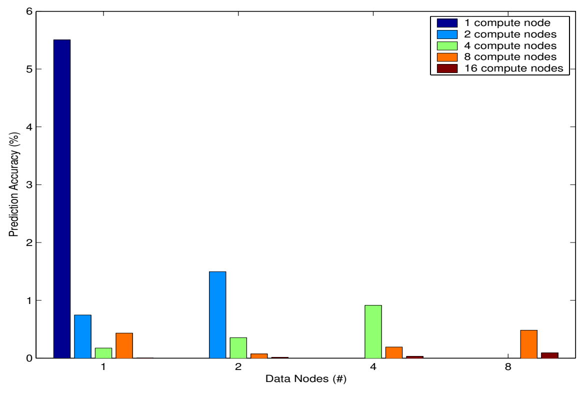

Figure 13: Prediction Errors for Vortex Detection on a Different Cluster, 1.85 GB dataset, Base profile: 1-1 with 710 MB

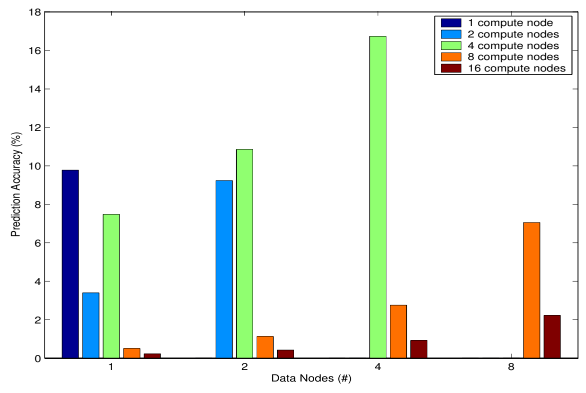

Figure 12: Prediction Errors for Molecular Defect Detection On a Different Cluster, 1.8 GB dataset, Base profile: 4-4 with 130 MB

The purpose of our final set of experiments is to show that our approach can successfully model application behavior on a cluster with different types of machines than the one where the base profile was collected on. We use a set of representative FREERIDE-G applications to compute average component-wise scaling factors between 2 clusters. Then, prediction is performed for applications which were not in this set. The base profile was collected on 700 MHz Pentium machines, and predictions were made for 2.4 GHz Opteron 250 machines.

In our first experiment, we evaluate prediction accuracy for the EM clustering application. Dataset size is 350 MB for base profile configuration and 700 MB for the configuration for which execution time is being predicted. Parallel configuration of the base profile is 8 data and 8 compute nodes. Kmeans clustering, kNN search and vortex detection applications' execution times with the base profile were used to compute component speedups between 2 clusters. It should be noted that scaling factors for the computation component did vary considerably across applications, ranging from 0.233 for kNN to 0.370 for Vortex detection.

The results are presented in Figure 11. Overall, the results are quite good, though inaccuracy levels are higher than in other experiments, as we would expect. Prediction errors are particularly higher for configurations using 8 compute nodes, which is also our base configuration. The reason for this is because of the difference in scaling factor across application. The average ratio we computed is 0.296, whereas the observed scaling factor for EM is 0.323. As the number of nodes is varied, we are able to accurately model parallel speedups, which reduces some of the prediction errors.

In Figure 12 we present accuracy resulting from execution time prediction for the molecular defect detection application. Dataset size is 130 MB for base profile configuration and 1.8 GB for the configuration for which execution time is being predicted. Parallel configuration of the base profile is 4 data nodes and 4 compute nodes. Again, Kmeans clustering, kNN search, and EM clustering applications' experimental results were used to compute component speedups between 2 clusters. The results with 4 compute nodes, which is the same number of nodes as in the base configuration, have the highest inaccuracy.

Figure 13 presents accuracy resulting from execution time prediction for the vortex detection application. Dataset size is 710 MB for base profile configuration and 1.85 GB for the configuration for which execution time is being predicted. Parallel configuration of the base profile is 1 data and 1 compute node. To compute component speedups between 2 clusters, kmeans clustering, kNN search, and EM clustering applications' experimental results were used. Like other predictions based on 1-1 configurations, largest inaccuracies are observed for predictions for which configurations have an equal number of compute and data nodes, so modeling different resources does not impact prediction accuracy. Since only one configuration has same number of compute nodes as the base profile configuration, the fact that this configuration results in the highest prediction error makes these results consistent with the rest of results in this subsection.

Overall, these results show that our approach for making predictions across platforms is reasonably accurate. FREERIDE-G applications do differ considerably in scaling of computations across different platforms, which results in some inaccuracy in predictions.

6. RELATED WORK

A significant amount of research has been performed in modeling communicating and/or parallel applications and predicting their performance to facilitate resource selection. Taura and Chien [33] have developed a heuristic algorithm that maps data-intensive communicating compute tasks onto heterogeneous resources, i.e., processors and links of various capacities. This approach tries to optimize throughput of a data-processing pipeline, taking both parallelism and communication volume and proximity into account. Snavely *et al.* [30] are investigating performance characterization of full parallel applications on large HPC systems. For this purpose, they have developed a framework that combines tools for gathering machine profiles and application signatures, providing automated convolutions of the two.

Communication characteristics of scientific applications on cluster architectures have been investigated by Vetter and Mueller [37]. Chien *et al.* have created a collection of simulation tools, called *MicroGrid* aimed at supporting systematic exploration of dynamic Grid software behavior [31, 5]. These simulation tools enable Globus applications to be run in arbitrary virtual grid environments, enabling opportunities for broad experimentation. Liu *et al.* have developed a general Resource Selection Service responsible for both selecting appropriate Grid resources based on a model presented to it as a part of the application submission process and mapping the application to the resources selected [21].

A project by Jang *et al.* [18] has presented a performance prediction module (Prophesy), to select resources for Pegasus based on previous performance history. Another project investigating execution time prediction of grid applications is Performance Analysis and Characterization Environment (PACE) [27]. PACE is structured around a hierarchy of performance models that describe the computing system in terms of its software, parallelization and hardware components.

The key distinction in our work is that focus on performance prediction for applications that fit into the processing structure of a particular middleware. This has allowed us to develop relatively simple models, which turn out to be very accurate for a number of applications.

A large amount of work has also been done in predicting individual components of a remote data analysis application, such as data transfer time over the network [8, 22, 35, 36]. As we stated earlier, this work can be incorporated as part of our framework, to allow us to predict performance over shared networks.

Several groups have also been developing support for grid-based data mining. One effort in this area is from Cannataro *et al.* [25, 26]. They present a structured Knowledge Grid tool-set for developing distributed data mining applications through workflow composition. Brezanny *et al.* [17, 2, 20] have also developed a GridMiner toolkit for creating, registering and composing datamining services into complex distributed and parallel workflows. Ghanem *et al.* [6, 9] have developed Discovery Net, an application layer for providing grid-based services allowing creation, deployment and management of complex data mining workflows. The goal of DataMiningGrid, carried out by Stankovski *et al.* [32], is to serve as a framework for distributed knowledge discovery on the grid.

There are significant differences between these efforts and our work. These systems do not offer a high-level interface for easing parallelization and abstracting remote data extraction and transfer. They also do not use detailed performance models for resource allocation.

7. CONCLUSIONS

This paper has addressed the problem of developing prediction models to be used for resource (and replica) selection in a grid middleware. By exploiting the fact that the processing structure of data mining and scientific data analysis applications developed on FREERIDE-G middleware involves generalized reductions, we are able to develop an accurate performance prediction model. We have evaluated our model using implementations of three well-known data mining algorithms and two scientific data analysis applications developed using FREERIDE-G. Results from these five applications show that we are able to accurately predict execution times for applications as we vary the number of storage nodes, number of nodes available for computation, the dataset size, the network bandwidth, and type of resource we are using.

8. REFERENCES

[1] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962 – 969, June 1996.

[2] P. Brezany, J. Hofer, A. Tjoa, and A. Wohrer. Gridminer: An infrastructure for data mining on computational grids. In *Proceedings of Australian Partnership for Advanced Computing Conference (APAC)*, Gold Coast, Australia, October 2003.

[3] C. Chang, A. Acharya, A. Sussman, and J. Saltz. T2: A customizable parallel database for multi-dimensional data. *ACM SIGMOD Record*, 27(1):58–66, March 1998.

[4] C. Chang, R. Ferreira, A. Acharya, A. Sussman, and J. Saltz. Infrastructure for building parallel database systems for multidimensional data. In *Proceedings of the Second Merged IPPS/SPDP (13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing)*. IEEE Computer Society Press, April 1999.

[5] Andrew A. Chien, Huaxia Xia, and Xin Liu. Network emulation tools for modeling grid behavior, December 11 2003.

[6] V. Curcin, M. Ghanem, Y. Guo, M. Kohler, A. Rowe, J. Syed, and P. Wendel. Grid knowledge discovery processes and an architecture for their composition. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 2002.

[7] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum Likelihood Estimation from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[8] Peter A. Dinda. Online prediction of the running time of tasks. In Sanjeev Setia, editor, *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS-01/PERFORMANCE-01)*, volume 29,1 of *ACM SIGMETRICS Performance Evaluation Review*, pages 336–337, New York, June 16–20 2001. ACMPress.

[9] M. Ghanem, Y. Guo, A. Rowe, and P. Wendel. Grid-based knowledge discovery services for high throghput informatics. In *The Eleventh IEEE International Symposium on High Performance Distributed Computing*, Edinburgh, Scotland, July 2002.

[10] Leo Glimcher and Gagan Agrawal. Parallelizing EM Clustering Algorithm on a Cluster of SMPs. In *Proceedings of Europar (to appear)*, 2004.

[11] Leo Glimcher, Gagan Agrawal, Sameep Mehta, Ruoming Jin, and Raghu Machiraju. Parallelizing a Defect Detection and Categorization Application. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[12] Leo Glimcher, Ruoming Jin, and Gagan Agrawal. FREERIDE-G: Supporting Applications that Mine Data Repositories. In *In proceedings of International Conference on Parallel Processing (ICPP)*, 2006.

[13] Leo Glimcher, Xuan Zhang, and Gagan Agrawal. Scaling and Parallelizing a Scientific Feature Mining Application Using a Cluster Middleware. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[14] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[15] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, (28):100–108, 1979.

[16] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[17] Ivan Janciak, Peter Brezany, and A. Min Tjoa. Towards the Wisdom Grid: Goals and Architecture. In *Proceedings of 4th International Conference on Parallel Processing and Applied Mathematics (PPAM)*, pages 796–803, 2003.

[18] S.-H. Jang, X. Wu, V. Taylor, G. Mehta, K. Vahi, and E. Deelman. Using performance prediction to allocate grid resources. Technical Report 2004-25, USC Information Sciences Institute and Texas A&M Department of Computer Science, November 2004.

[19] Ruoming Jin and Gagan Agrawal. Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2005.

[20] G. Kickinger, P.Brezany, A. Tjoa, and J. Hofer. Grid knowledge discovery processes and an architecture for their composition. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2004)*, Innsbruck, Austria, February 2004.

[21] Chuang Liu, Lingyun Yang, Ian T. Foster, and Dave Angulo. Design and evaluation of a resource selection framework for grid applications. In *HPDC*, pages 63–72, 2002.

[22] Dong Lu, Yi Qiao, Peter A. Dinda, and Fabian E. Bustamante. Characterizing and predicting tcp throughput on the wide area network. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 414–424, 2005.

[23] Dong Lu, Yi Qiao, Peter A. Dinda, and Fabián E. Bustamante. Modeling and taming parallel TCP on the wide area network. In *IPDPS*, 2005.

[24] R. Machiraju, J. Fowler, D. Thompson, B. Soni, and W. Schroeder. EVITA - Efficient Visualization and Interrogation of Terascale Datasets. In et al R. L. Grossman, editor, *Data Mining for Scientific and Engineering Applications*, pages 257–279. Kluwer Academic Publishers, 2001.

[25] M.Cannataro, A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio. Distributed Data Mining on Grids: Services, Tools, and Applications. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(6):2451–2465, December 2004.

[26] M.Cannataro and D. Talia. KNOWLEDGE GRID: An Architecture for Distributed Knowledge Discovery. *Communications of the ACM*, 46(1):89–93, January 2003.

[27] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE — A toolset for the performance prediction of parallel and distributed systems. *The International Journal of High Performance Computing Applications*, 14(3):228–251, Fall 2000.

[28] Yi Qiao, Jason Skicewicz, and Peter A. Dinda. An empirical study of the multiscale predictability of network traffic. In *HPDC*, pages 66–76, 2004.

[29] Sameep Mehta and Kaden Hazzard and Raghu Machiraju and Srinivasan Parthasarathy and John Willkins. Detection and Visualization of Anomalous Structures in Molecular Dynamics Simulation Data. In *IEEE Conference on Visualization*, 2004.

[30] Allan Snavely, Laura Carrington, Nicole Wolter, and The San. A framework for performance modeling and prediction, 2002.

[31] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, Kenjiro Taura, and Andrew A. Chien. The microgrid: a scientific tool for modeling computational grids. In *SC*, 2000.

[32] Vlado Stankovski, Michael May, Jürgen Franke, Assaf Schuster, Damian McCourt, and Werner Dubitzky. A service-centric perspective for data mining in complex problem solving environments. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 780–787, 2004.

[33] Kenjiro Taura and Andrew A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115, 2000.

[34] D.S. Thompson, R. Machiraju, M. Jiang, V. S. Dusi, J. Nair, and G. Craciun. Physics-Based Mining of Computational Fluid Dynamics Datasets. *IEEE Computational Science & Engineering*, 4(3), 2002.

[35] S. Vazhkudai, J. Schopf, and I. Foster. Predicting the performance of wide area data transfers. In *16th International Parallel and Distributed Processing Symposium (IPDPS '02 (IPPS & SPDP))*, page 34, Washington - Brussels - Tokyo, April 2002. IEEE.

[36] Sudharshan Vazhkudai and Jennifer M. Schopf. Predicting sporadic grid data transfers. In *HPDC*, page 188, 2002.

[37] J. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *16th International Parallel and Distributed Processing Symposium (IPDPS '02 (IPPS & SPDP))*, pages 27–29, Washington - Brussels - Tokyo, April 2002. IEEE.