High Efficient Scheduling Mechanism for Distributed Knowledge Discovery Platform

Meiqun Liu, Kun Gao
Culture and Communication College
Computer Science and Information Technology College
Zhejiang Wanli University
No. 8 South Qianhu Road, Ningbo 315100
China
http://www.zwu.edu.cn

Abstract: - Distributed data mining plays a crucial role in knowledge discovery in very large database. Since the distributed knowledge discovery process is both data and computational intensive, the Grid is a natural platform for deploying a high performance data mining service. The key issue for distributed data mining Grid system is how to scheduling data mining tasks in a high efficient way. In this paper, we propose a novel and efficient mechanism which is based on decomposing and mapping data mining tasks to DAG, and ordering them according the respective execution cost. The results show that this mechanism is scalable and feasibility.

Key-Words: - Grid Computing, Data Mining, Tasks Scheduling

1 Introduction

Distributed computing plays an important role in the Knowledge Discovery process for several reasons. On the one hand, Data Mining often requires huge amounts of resources in storage space and computation time. To make systems scalable, it is important to develop mechanisms that distribute the work load among several sites in a flexible way. On the other hand, data is often inherently distributed into several databases, making a centralized processing of this data very inefficient and prone to security risks. Distributed Data Mining explores techniques of how to apply Data Mining in a non-centralized way.

Data mining, which targets the goal of retrieving information automatically from large data sets, is one of the most important business intelligence technologies. Because of its high computational intensiveness and data intensiveness, data mining serves a good field of application for Grid technology.

The idea of data mining on the Grid is not new, but it has become a hot research topic only recently. The number of research efforts up to now is still quite limited (for a short summary see [1]). Many of the existing systems, such as NASA's Information Power Grid [2], TeraGrid [3] and Discovery Net [4] are either utilizing non-standard data mining techniques, or restricted to a special domain in the scientific realm.

The implementation of a Grid-based data mining system in the business realm will reveal the importance of specific requirements that are not so evident in the scientific realm, and will contribute to a generic design and implementation.

Advanced applications are continuing to generate ever larger amounts of valuable data, but we are in danger of being unable to extract fully the latent knowledge within the data because of insufficient technology [5]. The cost of producing this data is sometimes very high. Without effective ways to retrieve, analyze and manipulate it, this great expense will not yield the benefits to society that we might expect. Depending on the application area, the datasets mentioned above may include data produced by business transactions, medical investigations, scientific simulations, along with data obtained from satellites, telescopes, microscopes, seismic or tomographic techniques, etc. The volume of these datasets is already measured in terabytes and will soon total petabytes. They are often geographically distributed and their complexity is increasing, meaning that the extraction of meaningful knowledge requires more and more computing resources. The communities of users that need to access and analyze this data are often large and geographically distributed. This combination of large dataset size, geographic distribution of users and resources, and computationally intensive analysis results in complex and stringent performance demands that,

until recently, were not satisfied by any existing computational and data management infrastructure. Moreover, certain level of QoS and security has to be guaranteed for several classes of applications accessing the Grid data analysis services.

In tackling these problems, there are the beginnings of a new form of information technology known as the Computational Grid (or simply Grid) – an infrastructure that enables the integrated use of remote high-end computers, databases, scientific instruments, networks, and other resources. The fundamental challenge is to make Grid systems widely available and easy to use for a wide range of applications. This is crucial for accelerating their transition into fully operational environments. A significant amount of research for achieving these objectives has already started both in academia as well as industry. In the first phase, this research and development has been almost exclusively driven by "big science", like distributed high-performance simulations, high energy physics, material science, etc. Now the focus is going to shift to more general domains, closer to everyday life, like medical, business, and engineering applications [6]. However, to our best knowledge, till today, no relevant effort has been devoted to the development of Grid tools and services allowing to perform efficient knowledge discovery in large distributed databases and other datasets associated with these applications. Therefore, we have initiated a research effort to design and experimentally implement a novel infrastructure called GridMiner for knowledge discovery in databases coupled to Computational Grids.

The technology developed is being validated and tested on an advanced medical application addressing treatment of traumatic brain injury (TBI) victims. TBIs typically result from accidents in which the head strikes an object. The trajectory of the TBI patient management includes the following main points: trauma event (e.g. injury at a car or sport accident), first aid, transportation to hospital, acute hospital care, and home care. All these phases are associated with data collection into databases, which are currently autonomously managed by individual hospitals, and are, therefore, geographically distributed. Moreover, they are heterogeneous and need high data security precautions. The aim is to use the Grid technology for building a virtual organization, in which the cooperation of the participating hospitals and institutions is supported by the integration of the above databases. Knowledge discovered in these databases can help to significantly improve the quality of the decisions taken by the health care specialists involved in treatment of the TBI patients. In most situations, a nearly real-time response to knowledge discovery queries is urgently needed. It is obvious that this kind of applications introduces new challenges to the Grid developers.

The paper is organized as follows. Section 2 outlines the Concepts on Grid Computing that were followed in developing the current Grid technology, which we use as a basis for the Scheduler design. The kernel part, Sections 3 and 4, discusses how to decompose and map Data Mining Application to DAG, which is the main contribution of the paper. The architecture and concepts of the services developed for Grid databases access and data mediation are described in Section 4. Experimental Results are presented in Section 5. We briefly conclude and outline the future work in Section 6.

2 Concepts on Grid Computing

A Grid based computational infrastructure couples a vide variety of geographically distributed computational resources (such as PCs, workstations, and supercomputers), storage systems, data sources, databases, libraries, computational kernels, and special purpose scientific instruments, and presents them as a unified integrated resource which can be shared by communities ("virtual organizations") as they tackle common goals.

The early Grid efforts (the early to mid 1990s) started as projects to link supercomputing sites; at this time this approach was known as meta computing. The objective was to provide computational resources to a range of high performance applications. Today the grid infrastructure is capable of binding together more than just a few specialized supercomputing centers. It is more ubiquitous and can support diverse applications requiring large-scale computation and data. Essentially all major Grid projects are currently built on protocols and services provided by the Globus Toolkit (http://globus.org) that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. It provides the interoperability that is essential to achieve large-scale computation.

Grid computing (or the use of a computational grid) is the application of several computers to a single problem at the same time - usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. A well-known example of grid computing in the public domain is the ongoing SETI@Home (Search for Extraterrestrial Intelligence) project, in which thousands of people

share the unused processor cycles of their PCs in the search for signs of "rational" signals from outer space. According to John Patrick, IBM's vice-president for Internet strategies, "the next big thing will be grid computing."

Grid computing depends on software to divide and apportion pieces of a program among several computers, sometimes up to many thousands. Grid computing can also be thought of as distributed and large-scale cluster computing, as well as a form of network-distributed parallel processing. It can be small -- confined to a network of computer workstations within a corporation or it can be large -- a public collaboration across many companies or networks.

Grid computing is a form of distributed computing whereby a "super and virtual computer" is composed of a cluster of networked, loosely-coupled computers, acting in concert to perform very large tasks. This technology has been applied to computationally-intensive scientific, mathematical, and academic problems through volunteer computing, and it is used in commercial enterprises for such diverse applications as drug discovery, economic forecasting, seismic analysis, and back-office data processing in support of e-commerce and web services.

What distinguishes grid computing from conventional cluster computing systems is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Also, while a computing grid may be dedicated to a specialized application, it is often constructed with the aid of general purpose grid software libraries and middleware.

"Distributed" or "grid" computing in general is a special type of parallel computing which relies on complete computers (with onboard CPU, storage, power supply, network interface, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

The primary advantage of distributed computing is that each node can be purchased as commodity hardware, which when combined can produce similar computing resources to a multiprocessor supercomputer, but at lower cost. This is due to the economies of scale of producing commodity hardware, compared to the lower efficiency of designing and constructing a small number of custom supercomputers. The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections. This arrangement is thus well-suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors.

The high-end scalability of geographically dispersed grids is generally favorable, due to the low need for connectivity between nodes relative to the capacity of the public Internet.

There are also some differences in programming and deployment. It can be costly and difficult to write programs so that they can be run in the environment of a supercomputer, which may have a custom operating system, or require the program to address concurrency issues. If a problem can be adequately parallelized, a "thin" layer of "grid" infrastructure can allow conventional, standalone programs to run on multiple machines (but each given a different part of the same problem). This makes it possible to write and debug on a single conventional machine, and eliminates complications due to multiple instances of the same program running in the same shared memory and storage space at the same time.

As was already mentioned, Grid computing began with an emphasis on compute-intensive tasks, which benefit from massive parallelism for their computation needs, but are not data intensive; the data that they operate on does not scale in portion to the computation they perform. In recent years, this focus has shifted to more data-intensive applications, where significant processing is done on very large amounts of data.

Group developed a specification for a collection of OGSI-Compliant Grid database services. The first implementation of the service interfaces, OGSADAIS Release 1, is already available.

So far, only a little attention was devoted to knowledge discovery on the Grid [7, 8, 9], and to our best knowledge, no research effort about development of OGSA-based data mining services has been reported. Our solutions discussed in forthcoming sections can be viewed as a result of the natural evolution of parallel and distributed data mining technology and Grid Services and Grid Database Services development, and represent the first steps towards development standards for Open Service Architecture for data mining in Grid environments.

The application scenario described above reveals some important properties which advocate the adoption of a Grid-based data mining approach.

First, security is crucial for distributed data mining in enterprises. The business data are so sensitive that they should in no case be accessed without proper privileges.

Second, Grid-based data mining system for enterprises should be able to cope with the heterogeneity of not only computers, operating systems, networks, but also that from the different sources of data and different data mining software.

Third, in the highly competitive business environment, enterprises tend to be more flexible and dynamic. As a result, data mining systems in enterprises face stronger demand on the capability of processing data that are dynamically organized. In the franchise supermarket scenario, it often happens that new stores are opened for business expansion and non-profitable stores are closed.

With all above properties, a Grid-based approach becomes the natural choice, although there could be other similar techniques that are capable of achieving the same goal.

3 Decomposing Data Mining Application to DAG

K-Grid services can be used to construct complex Problem Solving Environments, which exploit DM kernels as basic software components that can be applied one after the other,her, in a modular way. A general DM task on the K-Grid can therefore be described as a Directed Acyclic Graph (DAG) whose nodes are the DM algorithms being applied, and the links represent data dependencies among the components. In this section, we present how to map data mining application to DAG.

3.1 Modeling Data Mining Applications

We surveyed three major classes of data mining applications, namely association rule mining, classification rule mining, and pattern discovery in combinatorial databases. We note the resemblance among the computation models of these three application classes.

A task is the main computation applied on a pattern. Not only are all tasks of any one application of the same kind, but tasks of different applications are actually very similar. They all take a pattern and a subset of the database and count the number of records in the subset that match the pattern. In the classification rule mining case, counts of matched records are divided into c baskets, where c is the number of distinct classes.

The similarities among the specifications of these applications are obvious, which inspired us to study the similarities among their computation models. They usually follow a generate-and-test paradigm-generate a candidate pattern, then test whether it is any good. Furthermore, there is some interdependence among the patterns that gives rise to pruning, i.e., if a pattern occurs too rarely, then so will any superpattern. These interdependences entail a lattice of patterns, which can be used to guide the computation.

In fact, this notion of pattern lattice can apply to any data mining application that follows this generate-and-test paradigm. We call this application class pattern lattice data mining. In order to characterize the computation models of these applications more concretely, we define them more carefully in Section 3.2.

3.2 Defining Data Mining Applications

1. A database D.

2. Patterns and a function len(pattern p) which non-negative integer. We use {} to represent zero-length patterns in association rule mining.

3. A function goodness(pattern p) which returns a measure of p according to the specifications of the application.

4. A function good(p) which returns 1 if p is a good pattern or a good subpattern and 0 otherwise. Zero-length patterns are always good.

The result of a data mining application is the set of all good patterns. If a pattern is not good, neither will any of its superpatterns be. In other words, it is necessary to consider a pattern if and only if all of its subpatterns are good.

Let us define an immediate subpattern of a pattern q to be a subpattern p of q where len(p) = len(q)-1. Conversely, q is called an immediate superpattern of p.

Except for the zero-length pattern, all the patterns in a data mining problem are generated from their immediate subpatterns. In order for all the patterns to be uniquely generated, a pattern q and one of its immediate subpatterns p have to establish a childparent relationship (i.e., q is a child pattern of p and p is the parent pattern of q). Except for the zero-length pattern, each pattern must have one and only one parent pattern. For example, in sequence pattern discovery, *FRR* can be a child pattern of *FR*; in association rule mining, {2, 3, 4} can be a child pattern of {2, 3}; and in classification rule mining, (C = c1)^(B = b2)^(A = a1) can be a child pattern of (C = c1)^(B = b2).

3.3 Solving Data Mining Applications

Having defined data mining applications as above, it is easy to see that an optimal sequential program

that solves a data mining application does the following:

1 generates all child patterns of the zero-length pattern;

2 computes goodness(p) if all of p's immediate subpatterns are good;

3 if good(p) then generate all child patterns of p;

4 applies 2 and 3 repeatedly until there are no more patterns to be considered.

Because the zero-length pattern is always good and the only immediate subpatterns of its children is the zero-length pattern itself, the computation starts on all its children, which are all length 1 patterns. After these patterns are computed, good patterns generate their child sets. Not all of these new patterns will be computed-only those whose every immediate subpattern is good will be.

3.4 Mapping data mining application to DAG

We propose to use a directed acyclic graph (dag) structure called exploration dag (E-dag, for short) to characterize pattern lattice data mining applications. We first describe how to map a data mining application to an E-dag.

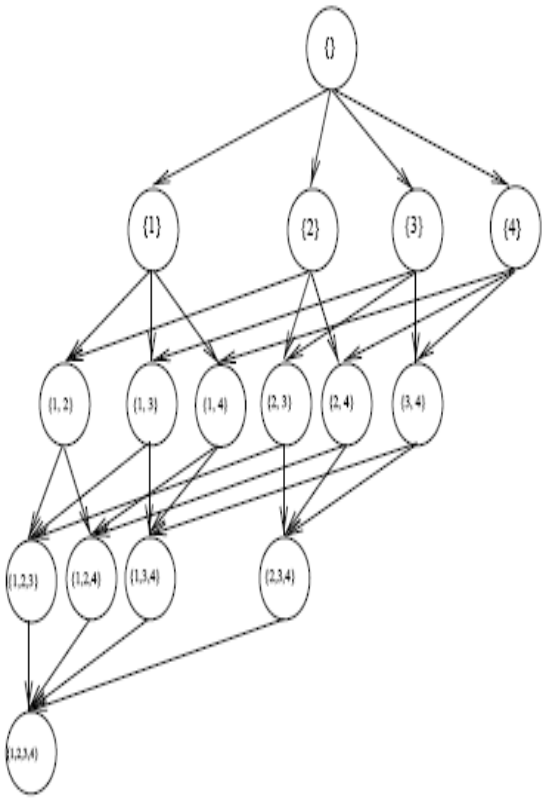

Fig. 1: A complete E-DAG for an association rule mining application on the set of items {1, 2, 3, 4}.

The E-dag constructed for a data mining application has as many vertices as the number of all possible patterns (including the zero-length pattern). Each vertex is labeled with a pattern and no two vertices are labeled with the same pattern. Hence there is a one-to-one relation between the set of vertices of the E-dag and the set of all possible patterns. Therefore, we refer to a vertex and the pattern it is labeled with interchangeably.

There is an incident edge on a pattern p from each immediate subpattern of p. All patterns except the zero-length pattern have at least one incident edge on them. The zero-length pattern has an outgoing edge to each pattern of length 1. Figure 1 shows an E-dag mapped from an association rule mining application.

4 Knowledge Grid Scheduler

4.1 Serialization Process

We consider that the basic building blocks of a DM task are algorithms and datasets. They can be combined in a structured way, thus forming a DAG. DM components correspond to a particular algorithm to be executed on a given dataset, provided a certain set of input parameters for the algorithm. We can therefore describe each DM components $\perp$ with the triple: $\perp$ = (A, D, {P}). Where A is the data mining algorithm, D is the input dataset, and {P} is the set of algorithm parameters. For example if A corresponds to "Association Mining", then {P} could be the minimum confidence for a discovered rule to be meaningful. It is important to notice that A does not refer to a specific implementation. We could therefore have more different implementations for the same algorithm, so that the scheduler should take into account a multiplicity of choices among different algorithms and different implementations. The best choice could be chosen considering the current system status, the programs availability and implementation compatibility with different architectures.

Scheduling DAGs on a distributed platform is a non-trivial problem which has been faced by a number of algorithms in the past. See [10] for a review of them. Although it is crucial to take into account data dependencies among the different components of the DAGs present in the system, we first want to concentrate ourselves on the cost model for DM tasks and on the problem of bringing communication costs into the scheduling policy. For

this reason, we introduce in the system an additional component that we call serializer (Figure 2), whose purpose is to decompose the tasks in the DAG into a series of independent tasks, and send them to the scheduler queue as soon as they become executable w.r.t. the DAG dependencies.

Such serialization process is not trivial at all and leaves many important problems opened, such as determine the best ordering among tasks in a DAG that preserve data dependencies and minimizes execution time.

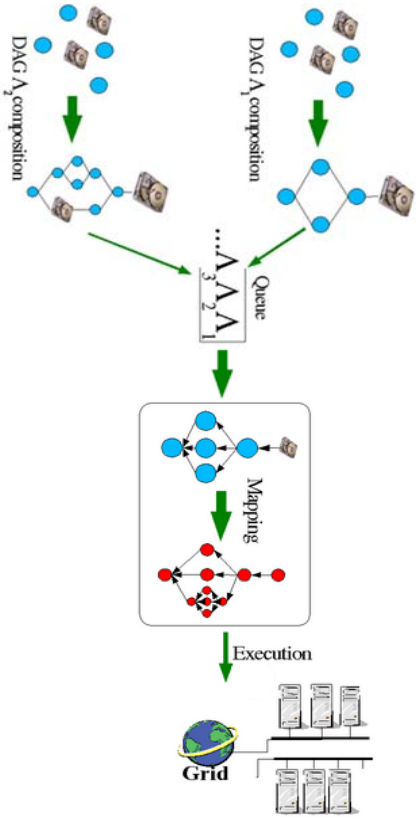

Fig. 2 Serializer

Nevertheless, at this stage of the analysis, we are mainly concerned with other aspects in the system, namely the definition of an accurate cost model for single DM tasks and the inclusion of communications into the scheduling policy.

4.2 Cost Model

The following cost model assumes that each input dataset is initially stored on a single machine m_h, while the knowledge model extracted must be moved to a machine m_k. Due to decisions taken by the scheduler, datasets may be moved to other machines and thus replicated, or may be partitioned among diverse machines composing a cluster for parallel execution. Therefore, the scheduler has to take into account that several copies (replicated or distributed) of a dataset may exist on the machines of its Grid.

Sequential execution. Suppose that the whole dataset is stored on a single machine m_h. Task t_i is executed sequentially by a code running on machine m_j, with an execution time of e_{ij} . In general we also have to consider the communications needed to move D_i from machine h to machine m_j, and the further communications to move the results

$|a_i(D_i)|$

to machine m_k. The total execution time is thus:

$$E_{ij} = |D_i| / b_{hj} + e_{ij} + |\alpha_i(D_i)| / b_{jk}$$

Of course, the relative communication costs involved in dataset movements are zeroed if either

h=j or j = k

Parallel execution. Task t_i is executed in parallel by a code running on a cluster c_{lJ} , with an execution time of e_{iJ}. In general, we have also to consider the communications needed to move D_i from machine m_h to cluster c_{lJ}, and to move the results

$|a_i(D_i)|$

to machine m_k. The total execution time is thus:

$$E_{iJ} = \sum_{m_t^l \in cl_J} \frac{|D_i| / |cl_J|}{b_{ht}} + e_{iJ} + \sum_{m_t^l \in cl_J} \frac{|\alpha_i(D_i)| / |cl_J|}{b_{tk}}$$

Of course, the relative communication costs are zeroed if the dataset is already distributed, and is allocated on the machines of c_{lJ}.

Performance metrics. E_{ij} and E_{iJ} are the expected total execution times of task t_i when no load is present in the system. When load is present on machines and networks, scheduling will delay the start and thus the completion of a task. In the following we will analyze the actual completion time of a task for the sequential case. A similar analysis could be done for the parallel case.

Let C_{ij} be the wall-clock time at which all communications and sequential computation involved in the execution of t_i complete. To define C_{ij} we need to define the starting times of communications and computation. Let s_{hj} be the start time of communication needed to move the input dataset from machine h to machine j, let s_j be the start time of the sequential execution of task t_i on machine j, and, finally, let s_{jk} be the start time of communication needed to move the knowledge result model extracted from machines j to machine k. From the above definitions:

$$C_{ij} = (s_{hj} + \frac{|D_i|}{b_{hj}}) + \delta_1 + e_{ij} + \delta_2 + \frac{|\alpha_i(D_i)|}{b_{jk}} = s_{hj} + E_{hj} + \delta_1 + \delta_2$$

Where

$$\delta_1 = s_j - (s_{hj} + \frac{|D_i|}{b_{hj}} \geq 0$$

And

$$\delta_2 = s_{jk} - (s_j + e_{ij}) \geq 0$$

So, if A_i is the arrival time of task t_i, and t_i is the only task in execution on the system, then the optimal completion time of the task on machine m_j is:

$$\overline{C_{ij}} = A_i + E_{ij}$$

Suppose that $m_{\bar{j}}$ is the specific machine chosen by our scheduling algorithm for executing a task t_i.

Let

$$C_i = C_{\bar{i}\bar{j}}$$

And

$$\overline{C_i} = \overline{C_{\bar{i}\bar{j}}}$$

Let T be the set of tasks to be scheduled. The makespan for the complete scheduling is defined as $\max_{t_i \in T}(C_i)$

And measures the overall thoughput of the system.

4.3 Predicting DM Tasks Execution Time

Data mining application computation times depend on many factors: data size, specific mining parameters provided by users and actual status of the Grid etc. Moreover, the correlations between the items present in the various transactions of a dataset largely influence the response times of data mining applications. Thus, predicting its performance becomes very difficult.

Our application runtime prediction algorithms operate on the principle that applications with similar characteristics have similar runtimes. Thus, we maintain a history of applications that have executed along with their respective runtimes. To estimate a given application's runtime, we identify similar applications in the history and then compute a statistical estimate of their runtimes. We use this as the predicted runtime.

The fundamental problem with this approach is the definition of similarity; diverse views exist on the criteria that make two applications similar. For instance, we can say that two applications are similar because the same user on the same machine submitted them or because they have the same application name and are required to operate on the same size data. Thus, we must develop techniques that can effectively identify similar applications. Such techniques must be able to accurately choose applications' attributes that best determine similarity. Having identified a similarity template, the next step is to estimate the applications' runtime based on previous, similar applications. We can use several statistical measures to compute the prediction, including measures of central tendency such as the mean and linear regression.

Rough sets theory as a mathematical tool to deal with uncertainty in data provides us with a sound theoretical basis to determine the properties that define similarity. Rough sets operate entirely on the basis of the data that is available in the history and require no external additional information. The history represents an information system in which the objects are the previous applications whose runtimes and other properties have been recorded. The attributes in the information system are these applications' properties. The decision attribute is the application runtime, and the other recorded properties constitute the condition attributes. This history model intuitively facilitates reasoning about the recorded properties so as to identify the dependency between the recorded attributes and the runtime. So, we can concretize similarity in terms of the condition attributes that are relevant and significant in determining the runtime. Thus, the set of attributes that have a strong dependency relation with the runtime can form a good similarity template.

The objective of similarity templates in application runtime estimation is to identify a set of characteristics on the basis of which we can compare applications. We could try identical matching, i.e. if n characteristics are recorded in the history, two applications are similar if they are identical with respect to all n properties. However, this considerably limits our ability to find similar applications because not all recorded properties are necessarily relevant in determining the runtime. Such an approach could also lead to errors, as applications that have important similarities might be considered dissimilar even if they differed in a characteristic that had little bearing on the runtime.

A similarity template should consist of the most important set of attributes that determine the runtime without any superfluous attributes. A reduct consists of the minimal set of condition attributes that have the same discerning power as the entire information system. In other words, the similarity template is equivalent to a reduct that includes the most significant attributes. Finding a reduct is similar to feature selection problem. All reducts of a dataset can be found by constructing a kind of discernibility function from the dataset and simplifying it.

For further detailed information see [11].

4.4 Scheduling Policy and Execution Model

We now describe how this cost model can be used by a scheduler that receives a list of jobs to be executed on the K-Grid, and has to decide for each of them which is the best resource to start the execution on.

Choosing the best resource implies the definition of a scheduling policy, targeted at the optimization of some metric. One frequent choice [12] is to minimize the completion time of each job. This is done by taking into account the actual ready time for the machine that will execute the job and the cost of execution on that machine, plus the communications needed. Therefore for each job, the scheduler will chose the machine that will finish the job earlier. For this reason in the following we refer to such policy as Minimum Completion Time (MCT).

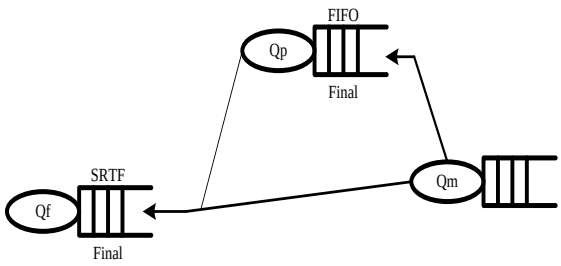

Fig. 3: The model of the scheduler.

Jobs $\lfloor$ (A, D, {P}) arrive at the scheduler from an external source, with given timestamps. They queue in the scheduler and wait. We assume the jobs have no dependencies among one another and their interarrival time is given by an exponential distribution.

The scheduler internal structure can be modeled as a network of three queues, plotted in Figure 3, with different policies.

Jobs arrive in the main queue Q_m, where predicting jobs are generated and appended to the predicting queue Q_p. Both queues are managed with a FIFO policy. From Q_p jobs are inserted into the system for execution. Once predicting is completed, the job in inserted in the final queue Q_f, where it is processed for the real execution. Since the scheduler knows the duration of jobs in Q_f, due to the prior predicting, Q_f is managed with a Shortest Remaining Time First (SRTF) policy in order to avoid light (interactive) jobs being blocked by heavier (batch) jobs.

The life-cycle of a job in the system is the following:

1. Jobs arrive in the main queue Q_m with a given timestamp. They are processed with FIFO policy. When a job is processed, the scheduler generates a predicting job and put this request in the predicting queue, with the same timestamp.

2. If a job in Q_m has parameters equals to that of a previously processed job, it is directly inserted into the final queue Q_f, with the same timestamp.

3. If the predicting job has finished, it is inserted in the Q_f queue, and timestamp given by the current time. Every time a job leaves the scheduler, a global execution plan is updated, that contains the busy times for every host in the system, obtained by the cost model associated to every execution.

4. Every time a job has finished we update the global execution plan.

5. When predicting is successfully finished, jobs are inserted in Q_f, where different possibilities are evaluated and the best option selected. Jobs in Q_f are processed in an SRFT fashion. Each job has an associated duration, obtained from the execution of predicting.

5 Some Preliminary Results

We adopted the MCT (Minimum Completion Time) [13] +rough set approach to validate that our hypothesis is feasible and efficient. The mapper does not consider node multitasking, and is responsible for choosing the schedule for computations involved in the execution of a given task, but also of starting tasks and checking their completion. The MCT mapping heuristics is very simple. Each time a task is submitted, the mapper evaluates the expected ready time of each machine. The expected ready time is an estimate of the ready time, the earliest time a given resource is ready after the execution of jobs previously assigned to it. Such estimate is based on both estimated and actual execution times of all the tasks that have been assigned to the resource in the past. To update resource ready times, when computations involved in the execution of a task complete, a report is sent to the mapper. The mapper then evaluate all possible execution plans for other task and chooses the one that reduce the completion time of the task. To evaluate our MCT scheduler that exploits rough set as a technique for performance prediction, we designed a simulation framework that allowed us to compare our approach with a Blind mapping strategy, which does not base its decisions on

performance predictions at all. Since the blind strategy is unaware of predicted runtime, so it scheduled tasks according the principle of FCFS (first come first serve).

time and data mining execution time on Grid. Finally, according the priori estimation of cost, we propose the method for tasks scheduling to minimize total response time in grid environment.

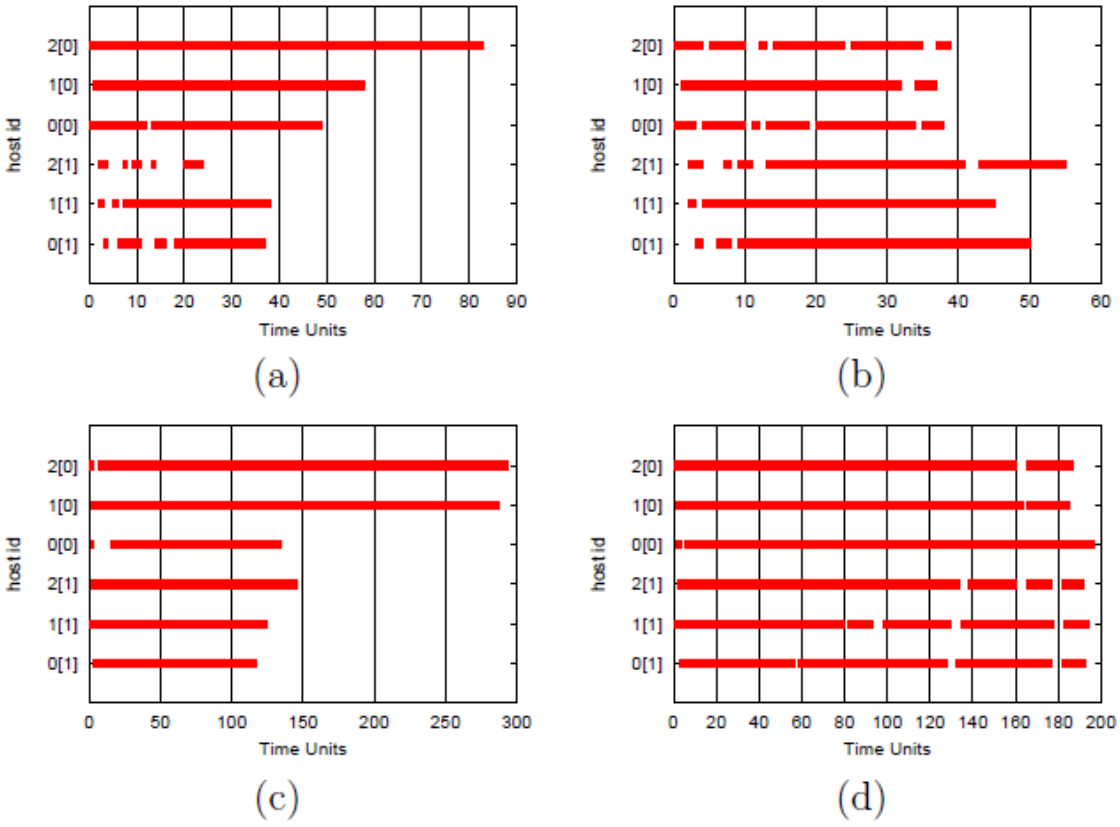

Fig. 4 Gannt charts showing the busy times (in time units of 100 sec.) of our six machines when either the 10% (a,b) or the 60% (c,d) of the tasks are expensive: (a,b) blind scheduling heuristics, (c,d) MCT+rough set scheduling

The simulated environment is composed of fifteen machines installed with GT3. Those machines have different physical configurations, operating systems and bandwidth of network. We used histories with 500 records as the condition attributes for estimation applications runtime. Data Ming tasks to be scheduled arrive in a burst, according to an exponential distribution, and have random execution costs. Datasets are all of medium size, and are randomly located on those machines. Figure 4 shows the improvements in makespans obtained by our technique over the blind one when the percentage of heavy tasks is varied.

6 Conclusion

We propose a new solution for data mining task scheduling in Grid environment. First, we propose map a data mining application to DAG. Then, we propose a cost model for predicting the data transfer

References:

[1] Cannataro, M.; Talia, D.: The Knowledge Grid, Communications of the ACM, Vol. 46, No. 1, pp89-93, Jan. 2003

[2] Hinke, T.; Novotny, J.: Data Mining on NASA's Information Power Grid, Proc. 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, Pennsylvania, Aug. 1-4, 2000

[3] Conover, H. et.al.: Data Mining on the TeraGrid, Poster Presentation, Supercomputing Conference 2003, Phoenix, AZ, Nov. 15, 2003

[4] Curcin, V. et.al.: Discovery Net: Towards a Grid of Knowledge Discovery, Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, pp658-663, 2002

[5] R. Williams et al. Large scientific databases. Joint EU-US Workshop, Annapolis, USA, September 1999.

[6] GRIDS: e-Science to e-Business. ERCIM News, April 2001.
[7] M. Cannataro, D. Talia, and P. Trunfio. Design of distributed data mining applications on the knowledge grid.
[8] M. Cannataro, D. Talia, and P. Trunfio. Distributed data mining on the grid. Future Generation Computer Systems, 18:1101–1112, 2002.
[9] R. Moore. Knowledge-Based Grids. Technical Report TR-2001-02, San Diego Supercomputer Center, January 2001.
[10] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing, 59(3):381–422, 1999.
[11] Kun Gao, Youquan Ji, Meiqun Liu, Jiaxun Chen, Rough Set Based Computation Times Estimation on Knowledge Grid, Lecture Notes in Computer Science, Volume 3470, July 2005, Pages 557 – 566.
[12] H. J. Siegel and A. Shoukat. Techniques for mapping tasks to machines in heterogeneous computing systems. Journal of Systems Architecture, 2000.
[13] Tracy D. Braun, Howard Jay Siegel, Noah Beck.: A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, Journal of Parallel and Distributed Computing, 2001