High Efficient Scheduler for Distributed Data Mining Applications

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. The key issue for distributed data mining systems 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 workload 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.

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 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.

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 wide 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 metacomputing. 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 problem 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.

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, 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.

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

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 $\text{len}(p) = \text{len}(q)-1$. 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, \((C = c1) \wedge (B = b2) \wedge (A = a1)\) can be a child pattern of \((C = c1) \wedge (B = b2)\).

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.

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.

![Diagram](https://via.placeholder.com/150)

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 \( L \) with the triple: \( L = (A, D, \{P\}) \). 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 [6] 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, 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.

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 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 [7] 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).

Choosing the best resource implies the definition of a scheduling policy, targeted at the optimization of some metric. One frequent choice [7] 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).

Choosing the best resource implies the definition of a scheduling policy, targeted at the optimization of some metric. One frequent choice [7] 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).

5 Some Preliminary Results

We adopted the MCT (Minimum Completion Time) [8] +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.
Fig. 3 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

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).

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 Mining 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 3 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 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.

References:
Distributed Computing, Pittsburgh, Pennsylvania, Aug. 1-4, 2000


