A Sampling-based Method for Dynamic Scheduling in Distributed Data Mining Environment

Jifang Li
Computer Science and Information Technology College
Zhejiang Wanli University
P.R.China
http://www.zwu.edu.cn

Abstract: - In this paper, we propose a new solution for dynamic task scheduling in distributed environment. The key issue for scheduling tasks is that we can not obtain the execution time of irregular computations in advance. For this reason, we propose a method which is based on sampling to some typical data mining algorithm. We argue that a function is existed in the items: execution time, the size of data and the algorithm, therefore we can deduce the execution time of a data mining task from the corresponding the size of data and algorithm. The experimental results show that almost all the algorithms exhibits quasi linear scalability, but the slope of different algorithms is different. We adopt this sampling method for process the tasks scheduling in distributed data mining environment. The experimental results also show the sampling method is applicable to task scheduling in dynamic environment and can be adopted to obtain a higher result.

Key-Words: - Sampling, Data Mining, Distributed Computing

1 Introduction

Dynamic distributed environment, especially Grid computing or the use of a computational grid, is applying the resources of many computers in a network 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.

Grid computing requires the use of software that can divide and farm out pieces of a program to as many as several thousand computers. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. It can be confined to the network of computer workstations within a corporation or it can be a public collaboration, in which case it is also sometimes known as a form of peer-to-peer computing.

A number of corporations, professional groups, university consortiums, and other groups have developed or are developing frameworks and software for managing grid computing projects. The European Community (EU) is sponsoring a project for a grid for high-energy physics, earth observation, and biology applications. In the United States, the National Technology Grid is prototyping a computational grid for infrastructure and an access grid for people. Sun Microsystems offers Grid Engine software. Described as a distributed resource management (DRM) tool, Grid Engine allows engineers at companies like Sony and Synopsys to pool the computer cycles on up to 80 workstations at a time. At this scale, grid computing can be seen as a more extreme case of load balancing.

Grid computing appears to be a promising trend for three reasons:

(1) Its ability to make more cost-effective use of a given amount of computer resources

(2) As a way to solve problems that can't be approached without an enormous amount of computing power, and

(3) Because it suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as collaboration toward a common objective.

In some grid computing systems, the computers may collaborate rather than being directed by one managing computer. One likely area for the use of grid computing will be pervasive computing applications - those in which computers pervade our environment without our necessary awareness.

Thanks to vast improvements in wide-area network performance and powerful yet low-cost computers, Grid computing has emerged as a promising attractive computing paradigm. Computational Grids aim to aggregate the power of heterogeneous, geo-graphically distributed, multiple-domain-spanning computational resources to provide high performance or high-throughput computing. To achieve the promising potentials of computational

Grids, an effective and efficient scheduling system is fundamentally important.

Scheduling systems for traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Scheduling in Grid environments is significantly complicated by the unique characteristics of Grids.

Task scheduling is an important aspect of dynamic distributed environment. Most of the heuristics for this NP-hard problem are based on a very simple system model of the target distributed system. Experiments revealed the inappropriateness of this classic model to obtain accurate and efficient schedules for real-systems. In order to overcome this shortcoming, a new scheduling model was proposed that considers the relationship among some important items. Even though the accuracy and efficiency improved with the consideration of the relationship, the existed method for predicting is still not good enough. The crucial aspect is the predicting the execution time of tasks. This paper investigates the relationship among data size, algorithm and execution time. The challenges for the scheduling techniques are analyzed and a new method for tasks scheduling is proposed based on sampling. Experiments on dynamic distributed environment show the significantly improved accuracy and efficiency of the new methods.

2 Problem Formulation

A scheduler is the mediate resource manager as the interface between the consumers and the underlying resources. Scheduling is a core function of resource management systems.

In a distributed environment, on one hand, there is a suite of computational resources interconnected by networks; on the other hand, there is a group of users who will submit applications for execution on the suite of resources. The scheduling system of such a distributed computing environment is responsible for managing the suite of resources and dealing with the set of applications. In face of a set of applications waiting of execution, the scheduling system should be able to allocate appropriate resources to applications, attempting to achieve some performance goals.

In traditional parallel computing environments, the scheduling system is made much simpler due to the uniform characteristics of both the target applications and the underlying resources. However, a computational Grid has more diverse resources as well as more diverse applications.

According to GGF's Grid scheduling dictionary [1], the Grid scheduler is responsible for:

(1) Discovering available resources for an application

(2) Selecting the appropriate system(s), and

(3) Submitting the application.

In brief, Grid scheduling is a software framework with which the scheduler collects resource state information, selects appropriate resources, predicts the potential performance for each candidate schedule, and determines the best schedule for the applications to be executed on a Grid system subject to some performance goals.

In principle, scheduling in Grids means two things: ordering and mapping. When there are more than one applications waiting for execution, ordering is performed in order to determine by which order the pending applications are arranged. Ordering is necessary if applications with priority or deadline are involved. Mapping is the process of selecting a set of appropriate resources and allocating the set of re-sources to the applications. For each mapping, the performance potential is estimated in order to decide the best schedule.

In general, a scheduling system of Grid computing environments aims at delivering better performance. Desirable performance goals of Grid scheduling includes: maximizing system throughput [2], maximiz-ing resource utilization, minimizing the execution time [3] and fulfilling economical

3 Related Works

To date, there have been a number of exciting initial efforts at developing scheduling systems for Grid environments. In this section, we focus on a representative group of these pioneering efforts to illustrate the state-of-the-art in Grid schedulers. It is often difficult to make comparisons between distinct efforts because each scheduler is usually developed for a particular system environment with different assump-tions and constraints. In the following section, I attempt to outline the features of each system and sum-marize their advantages and drawbacks. Also how does they fit the taxonomy.

3.1 Information Collection Systems

The information service infrastructure plays a particularly important role in a scheduling system. Meta Director Service (MDS) from Globus and Network Weather Service (NWS) are two popular systems serving to provide the information publication and collection of resources in a grid system.

3.1.1 MDS (Meta Directory Service)

The Globus Metacomputing Directory Service (MDS) [4] provides is a LDAP-based information service infrastructure for computational Grids. It is used to collect and publish status information of Grid resources. Examples of the information that can be retrieved from an MDS server include operating sys-tem type, processor type and speed, number of processors, and available memory size of resources.

Each GRIS (Grid Resource Information Service) is responsible for monitoring one resource and keeping all the current configuration, capabilities, and status of the resource. A GRIS can be queried for the information of a local resource. Each GIIS provides a means of aggregating information from several GRISs to present an information pool for a set of resources. Different GIIS can be further aggregated to manage more integrated information of a larger set of resources.

Thus, MDS provides a hierarchical method for providing information service in a Grid system, which is scalable and efficient. As a limitation, MDS can only provide current state information of the resources.

3.1.2 NWS (Network Weather Service)

The Network Weather Service (NWS) [5] is a distributed system that monitors and forecasts the performance of network and computation resources. Examples of the information that can typically be retrieved from an NWS server include the fraction of CPU available to a newly started process, the avail-able memory size, and the bandwidth with which data can be sent to a remote host.

Each resource has one NWS agent, which consists of three components: sensory subsystem, forecasting subsystem and reporting interface. NWS sensors collect the current performance that a resource is able to deliver at present. The collected performance data from sensors is input to the forecasting subsystem, and based on the forecasting models, different levels of performance in a future timeslot are forecast. Through the report interface, the predicted resource performance can be retrieved.

To avoid becoming a bottleneck of a system, the NWS does not have a centralized node that has the overall information of all the resources. The responsibility of information service is distributed to every resource. Superior to MDS, NWS provides not only the current performance information, but also the predicted performance information for a future timeslot which is more beneficial for estimating the performance behavior of a job running on the resource.

3.2 Condor

Condor [6] aims to increase utilization of workstation by hunting idle workstations for sharing job execution.

The condor Scheduling Structure Condor follows an approach to scheduling that lies between the centralized and decentralized scheme. For information collection, on one hand, Condor employs a Coordinator responsible for managing the set of available idle workstations. For scheduling jobs, on the other hand, each workstation itself is responsible for maintaining the local queue of jobs to be run and scheduling the jobs onto idle workstations for execution.

Condor is capable of check pointing and job migrating, which are important features for rescheduling. In Condor, a remote job can run on a workstation only when the workstation is idle, that is, the work-station has no local workload. Once a workstation has its own workload, the remote job currently running on the workstation is preempted. With the help of check pointing, the preempted job can be rescheduled to another idle workstation to resume its job, such that the previously accomplished results can be utilized.

The performance goal of Condor is to maximizing the throughput of the system, which is sys-tem-centric. The target applications of Condor are independent, non-real time batch jobs. The underlying resources are homogeneous, preemptive, non-dedicated, and non-time-shared. The description of re-sources is coarse-grained since only the availability of workstations is considered.

Condor supports site autonomy. However, communication overhead of transferring a job is not considered. It is only suitable for WAN-based environment.

3.3 Condor-G

A newly proposed version of Condor, Condor-G [7], leverages the advantages of both condor and Globus Toolkit [8] [9]. Globus Toolkit is a software infrastructure for setting up a Grid environment across multiple administrative domains, which sup-ports resources management, secure file transfer, information discovery and secure authentication and authorization in such a Grid environment. Based on Condor, Condor-G makes use of the mechanisms provided by Globus Toolkit to cross the boundaries of real institutions, aiming at utilizing the idle work-stations among these institutions. Also the job creation, job monitoring and result collection are heavily relied on the GRAM (Grid Resource Access Management) component of Globus.

In condor-G, each Job Submission Machine constructs a GridManager locally which manages local jobs, retrieves the available resources, and schedules the jobs onto the feasible resources. GSI (Grid Se-curity Infrastucture) mechanism of Globus is used by GridManager to do authentication and authorization with remote resources. Information collection of resources is based on MDS (Meta Directory Service) mechanism of Globus, which is in principle centralized.

The problem of scheduling a set of dependent jobs is solved by Condor-G by designing the local Grid Manager with the coordinating function. Thus the applications running on Condor-G is more fine-grained compared to that of Condor.

Resource heterogeneity is allowed in Condor through deploying the standard resource manager on re-sources. The computational resources in Condor could vary from workstations to clusters. But the description of resources remains coarse-grained.

Similar with Condor, Conder-G has the performance goal as maximizing the utilization of resources. Condor-G is resource-fault tolerant, meaning that it is able to cope with the resource failure. Condor-G allows inter-domain operation on remote resources that require authentication.

3.4 AppLeS

AppLeS [10] is an agent-based scheduling system for Grid environments, which targets to promote the performance of every individual application. AppLeS is based on the application-level scheduling paradigm in which everything in the system is evaluated in terms of its impact on the application. For each application, its performance goal is specified by the application itself.

Each Grid application has its own scheduler which determines and "actuates" a schedule. The schedule is computed based on the application characteristics, performance goal, and resources currently available to the application.

An AppLeS agent is organized in terms of four subsystems and a single active component called the Coordinator. The four subsystems are:

- The Resource Selector, which chooses and filters different resource combinations for the application's execution.

- The Planner, which generates a resource-dependent schedule for a given resource combination.

- The Performance Estimator, which generates a performance estimation for candidate schedules ac-cording to the user's performance metric, and

- The Actuator, which implements the 'best' schedule on the target resource management systems.

The Network Weather Service (NWS) is used to provide dynamic information of resources and predict the resource load for the time frame in which the application will be scheduled. The User Interface pro-vides specific information about the structure, characteristics and current implementations of the applica-tion and its jobs, as well as information on the user's criteria for performance, execution constraints, and preferences. Finally, Models provide a repository of default application class models and applica-tion-specific models to be used for performance estimation.

The target class of applications in AppLeS has a common structure: master/slave. The execution time model of the class of applications can be expressed as follow:

ExecTime = MasterComp + maxi { SlaveCompi } + ResultGather

Where MasterComp, SlaveCompi and ResultGather provide a decomposition of the application execution behavior. AppLeS seems not to solve the multi-domain problem. Non-dedicated, time-shared re-sources are involved in AppLeS. In AppLeS, the performance goal is determined by the application itself. It is achieved through considering the application profile when selecting resources and making scheduling decision.

AppLeS employs NWS as its information service provider, which has a decentralized organization. Since each application has its own scheduler, it is obvious that the scheduler organization of AppLeS is evenly decentralized.

It is not difficult to note that there will be many AppLeS in a system simultaneously, each working on behalf on its own application. A worst-case scenario is that all of the AppLeS may identify the same re-sources as "best" for their applications and seek to use them simultaneously. Recognizing that the targeted resources are no longer optimal or available, they all might seek to reschedule their applications on another resource.

3.5 Legion

The Legion project [11] [12] from the University of Virginia is an object-based Grid environment, in-tended to connect a large suite of wide-area computational resources, with the illusion of providing a single virtual machine.

Legion is an object-oriented system consisting of independent disjoint objects that communicate with one another via method invocation. Classes

define the types of their instances. An object is an instance of a class, which is responsible for managing a single resource. For example, HostClass encapsulates ma-chine capabilities, e.g., CPU capability and memory size.

In Legion, each application is typically scheduled by a customized scheduler associated to it. An ap-plication is basically also an object, and this object is to be instantiated into several instances. The sched-uler is responsible for selecting a set of appropriate execution machines and mapping the set of instances onto the set of selected machines.

The Collection acts as a repository for information describing the state of the resources comprising the system. The Scheduler computes the mapping of instances in a class to resources. At a minimum, the Scheduler knows how many instances of each class must be started. The enactor involves implementing a schedule for a class forwarded from the scheduler.

Legion's targeted applications can be diverse since each application can develop its own scheduler. Due to the fact that the scheduler well knows the application-specific knowledge, the scheduler is able to produce efficient schedules for the application. The resources that Legion wants to utilize can be also het-erogeneous. But it seems that Legion does not incorporate the dynamics in network behavior, which will lead to a drawback. Legion favors the performance goal of minimizing the execution time of an individual application.

Legion employs a centralized entity for information collection whilst it uses the decentralized scheme of scheduler organization. Through job monitoring, Legion is capable of rescheduling.

3.6 Nimrod-G

Nimrod [13] is a parametric study system, which uses a simple declarative parametric modeling language for expressing a parametric experiment. It has worked successful with a static set of computational resources, but is unsuitable as implemented in the large scale dynamic context of computational Grids, where resources are scattered across several domains.

To overcome that shortcoming, the Nimrod/G [14] makes use of the Globus toolkits for dynamic resource discovery and dispatches jobs over computational Grids. Nimrod/G supports an integrated computational economy in its scheduling system. This means that Nimrod/G can schedule applications on the basis of deadlines and budget.

In Nimrod/G, each application has one program and a large set of independent parameters to be studied, and hence it has a large number of independent jobs. An application specifies a deadline by which the application is expected to complete, and a price which the application owner will pay for the completion of the application. Each computational resource is specified a cost which the consumer should pay in order to use the resource. Briefly, a parametric study application is performed by Nimrod/G through the following steps:

1. Discovery: First the number and then the identity of the lowest-cost set of resources able to meet the deadline are identified. A cost matrix is used to identify low-cost resources; queries to the MDS directory service are then used to determine resource. The output from this phase is a set of resources to which jobs should be submitted, ordered by the cost to the application.

2. Allocation: Jobs are allocated to the candidate resources identified in Step 1.

3. Monitoring: The completion time of submitted jobs is monitored, hence establishing an execu-tion rate for each resource.

4. Refinement: Rate information is used to update estimates of typical execution times on different resources and hence the expected completion time of the job. This refinement process may lead us to return to 1.

The scheme continues until the deadline is met, or the cost budget is exceeded. If the latter occurs, the user is advised and the deadline can be modified accordingly.

In Nimrod/G, the description of each application is coarse-grained. The targeted applications in Nimrod/G are specified with deadline and it is possible the deadline may not be met, and therefore they are soft real-time applications. The description of each resource is also coarse-grained. These resources are non-dedicated, timeshared and across multiple administrative domains.

Nimrod/G uses a hierarchical scheme of information service and a decentralized scheme of scheduler organization. Nimrod/G is useful for parametric study applications. Thus, the classes of applications sup-ported are limited. It should be noted that in order to make the economy-based scheduling mechanism practically work, much work is still to be done.

In the new applications of Grid computing, some significative efforts were present in [19][20][21][22]. Authors proposed a new framework for knowledge discovery based on Grid Computing. Some similar NP-Complete problem appeared in the architecture. Authors proposed some novel solution based on

rough set for solving the NP-Complete problem. Rough set theory can provide us a sound solution.

4 Cost Model

In the following cost model we assume that each input dataset is initially stored on at least a single machine mh, while the knowledge model extracted must be moved to a machine mk. Due to decisions taken by the scheduler, datasets may be replicated onto other machines, or partitioned among the machines composing a cluster.

Sequential execution. Dataset D_i is stored on a single machine m_h. Task t_i is sequentially executed on machine m_j, and its execution time is e_{ij}. The knowledge model extracted $|a_i(D_i)|$ must be returned to machine m_k. We have to consider the communications needed to move D_i from m_h to m_j , and those to move the results to m_k. Of course, the relative communication costs involved in dataset movements are zeroed if either h = j or j = k. The total execution time is thus:

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

Parallel execution. Task t_i is executed in parallel on a cluster c_{lJ} , with an execution time of e_{iJ}. In general, we have also to consider the communications needed to move and partition D_i from machine m_h to cluster c_{lJ} , and to return the results $|a_i(D_i)|$ to machine m_k. Of course, the relative communication costs are zeroed if the dataset is already distributed, and is allocated on the machines of c_{lJ}. The total execution time is thus:

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

Finally, consider that the parallel algorithm we are considering requires coallocation and coscheduling of all the machines of the cluster. A different model of performance should be used if we adopted a more asynchronous distributed DM algorithm, where first independent computations are performed on distinct dataset partitions, and then the various results of distributed mining analysis are collected and combined to obtain the final results.

To optimize scheduling, our mapper has to forecast the completion time of tasks. To this end, the mapper has also to consider the tasks that were previously scheduled, and that are still queued or running. Therefore, in the following we 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 on machine m_j complete. To derive C_{ij} we need to define the starting times of communications and computation on the basis of the ready times of interconnection links and machines. Let s_{hj} be the starting time of the communication needed to move D_i from m_h to m_j , s_j the starting time of the sequential execution of task t_i on m_j, finally, s_{jk} the starting time of the communication needed to move $a_i(D_i)$ from m_j to m_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 $\delta_2 = s_{jk} - (s_j + e_{ij}) \geq 0$

If m_j is the specific machine chosen by our scheduling algorithm for executing a task t_i, where T is the set of all the tasks to be scheduled, we define

$C_i = C_{\bar{ij}}$.

The makespan for the complete scheduling is thus defined as

$\max_{t_i \in T}(C_i)$

and its minimization roughly corresponds to the maximization of the system thoughput.

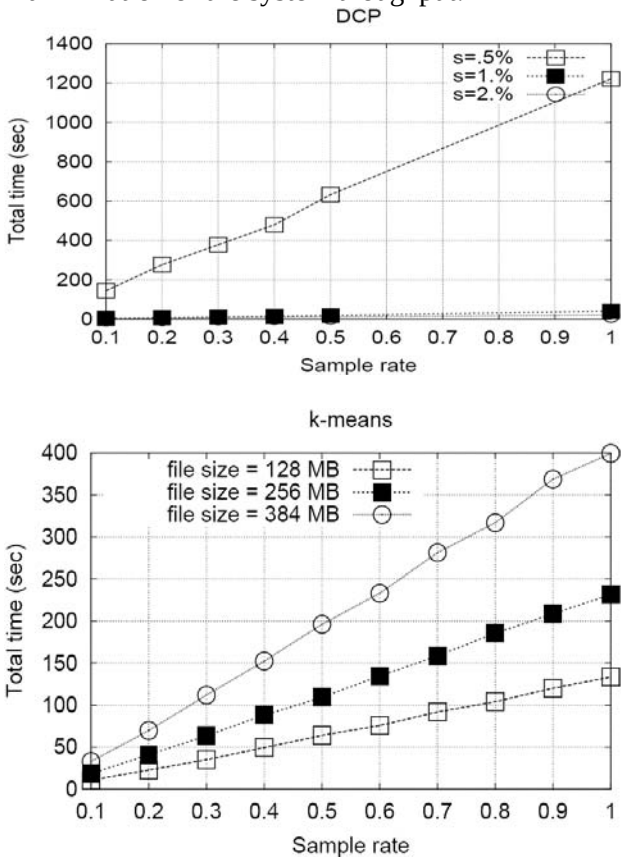

Fig.1 Execution time of the DCP ARM algorithm (a), and the k-means clustering one (b), as a function of the sample rate of the input dataset.

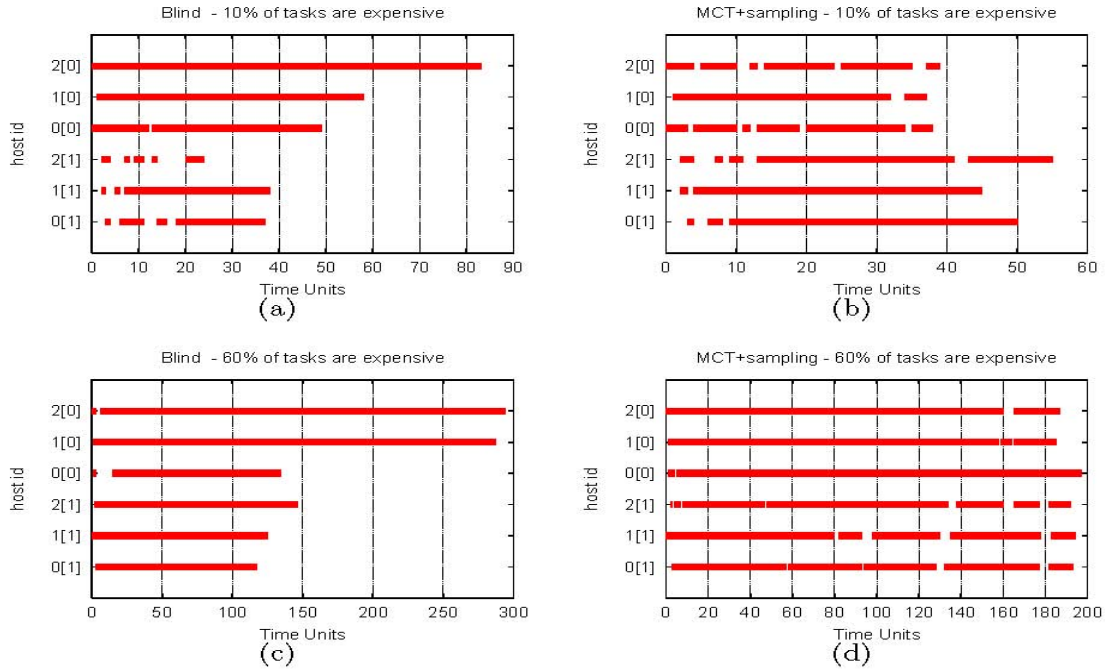

Fig. 2 Gannt charts showing the busy times (in time units of 100 sec.) of our six machines when either the 10% (a,b) or the 60% (c,d) of the tasks are expensive: (a,b) blind scheduling heuristics, (c,d) MCT+sampling

5 Sampling method

The rationale of our approach is that, since DM tasks may be very expensive, it may be more profitable to spend a small additional time to sample their execution in order to estimate performances and schedule tasks more accurately, than adopting a blind scheduling strategy.

For example, is a task is guessed to be expensive, we may be profitable to move data to execute the task on a remote machine characterized by an early ready time, or distribute data on a cluster to perform the task in parallel. Differently from [15], we are not interested in the accuracy of the knowledge extracted from a sampled dataset, but only in an approximate performance prediction of the task. To this end, it becomes important to study and analyze memory requirements and completion times of a DM algorithm as a function of the size of the sample exploited, i.e. to study the scalability of the algorithm. From this scalability study we expect to derive, for each algorithm, functions that, given the measures obtained with sampling, return predicted execution time and memory requirement for running the same analysis on the whole dataset.

Suppose that a given task t_i is first executed on a sample D'_i of dataset D_i on machine m_j. Let e'_{ij} be this execution time, and let $e'_i=e_{ij}/p_j$ be the normalized execution time on the sample. Sampling is feasible as a method to predict performance of task t_i iff, on the basis of the results of sampling, we can derive a cost function F(), such that $e_i = F(|D_i|)$. In particular, the coefficients of F() must be derived on the basis of the sampled execution, i.e., in terms of e_i', D_i', and $|D_i'|$. The simplest case is when the algorithm scales linearly, so that F() is a linear function of the size of the dataset, i.e.

$$e_i = \gamma |D_i|$$

Where $\gamma = e_i'/|D_i'|$

We analyzed two DM algorithms: DCP, an ARM algorithm which exploits out-of-core techniques to enhance scalability [16], and k-means, the popular clustering algorithm. We ran DCP and k-means on synthetic datasets by varying the size of the sample considered. The results of the experiments are promising: both DCP and k-means exhibit quasi linear scalability with respect to the size of the sample of a given dataset, when user parameters are fixed. Figure 1 (a) reports the DCP completion times on a dataset of medium size (about 40 MB) as a function of the size of the sample, for different user parameters (namely the minimum support s% of frequent itemsets). Similarly, in Figure 1. (b) the completion time of k-means is reported for different datasets, but for identical user parameters (i.e., the number k of clusters to look for). The results obtained for other datasets and other user parameters are similar, and are not reported here for sake of brevity. Note that the slopes of the various linear curves depend on both

the specific user parameters and the features of the input dataset D_i. Therefore, given a dataset and the parameters for executing one of these DM algorithms, the slope of each curve can be captured by running the same algorithm on a smaller sampled dataset D_i'. For other algorithms, scalability curves may be more complex than a simple linear one. For example when the dataset size has a strong impact on the in-core or out-cores behavior of an algorithm, or on the main memory occupation. So, in order to derive an accurate performance model for a given algorithm, it should be important to perform an off-line training of the model, for different dataset characteristics and different parameter sets.

Another problem that may occur in some DM algorithms, is the generation of false patterns for small sampling sizes. In fact, according to [17], we found that the performance estimation for very small sampling sizes may overestimate the actual execution times on the complete datasets. An open question is to understand the impact of this overestimation in our Grid scheduling environment.

4 On-line Scheduling of DM Tasks

We analyzed the effectiveness of a centralized on-line mapper based on the MCT (Minimum Completion Time) heuristics [18], which schedules DM tasks on a small organization of a K-Grid. The mapper does not consider node multitasking, is responsible for scheduling both dataset transfers and computations involved in the execution of a given task t_i, and also is informed about their completions. The MCT mapping heuristics adopted is very simple. Each time a task t_i is submitted, the mapper evaluates the expected ready time of each machine and communication links. The expected ready time is an estimate of the ready time, the earliest time a given resource is ready after the completion of the jobs previously assigned to it. On the basis of the expected ready times, our mapper evaluates all possible assignment of t_i, and chooses the one that reduces the completion time of the task.

Note that 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 data transfers or computations involved in the execution of t_i complete, a report is sent to the mapper. Note that any MCT mapper can take correct scheduling decisions only if the expected execution time of a task is known. When no performance prediction is available for t_i, our mapper first generates and schedules t_i', i.e. the task t_i executed on the sampled dataset D_i'. Unfortunately, the expected execution time of sampled task t_i' is unknown, so that the mapper has to assume that it is equal to a given small constant. Since our MCT mapper can not be able to optimize the assignment of t_i', it simply assigns t_i' to the machine that hosts the corresponding input dataset, so that no data transfers are involved in the execution of t_i'. When t_i' completes, the mapper is informed about its execution time. On the basis of this knowledge, it can predict the performance of the actual task t_i, and optimize its subsequent mapping and scheduling.

5 Simulation Framework and Some Preliminary Results

We designed a simulation framework to evaluate our MCT on-line scheduler, which exploits sampling as a technique for performance prediction. We thus compared our MCT+sampling strategy with a blind mapping strategy. Since the blind strategy is unaware of actual execution costs, it can only try to minimize data transfer costs, and thus always maps each task on the machine that holds the corresponding input dataset. Moreover, it can not evaluate the profitability of parallel executions, so that sequential implementations are always preferred.

The simulated environment is similar to an actual Grid environment we have at disposal, and is composed of two clusters of three machines. Each cluster is interconnected by a switched fast Ethernet, while a slow WAN interconnection exists between the two clusters. The two clusters are homogeneous, but the machines of one cluster are two times faster than the machines of the other one. To fix simulation parameters, we actually measured average bandwidths b_{WAN} and b_{LAN} of the WAN and LAN interconnections, respectively. Unfortunately, the WAN interconnection is characterized by long latency, so that, due to the TCP default window size, single connections are not able to saturate the actual bandwidth available. This effect is exacerbated by some packet losses, so that retransmissions are necessary and the TCP pipeline can not be filled. Under these hypotheses, we can open a limited number of concurrent sockets, each one characterized by a similar average bandwidth bWAN (100KB/s).

We assumed that DM tasks to be scheduled arrive in a burst, according to an exponential distribution. They have random execution costs, but the x% of them corresponds to expensive tasks (1000 sec. as mean sequential execution time on the slowest machine), while the (100 − x)% of them are

cheap tasks (50 sec. as mean sequential execution time on the slowest machine). Datasets Di are all of medium size (50MB), and are randomly located on the machines belonging to the two clusters.

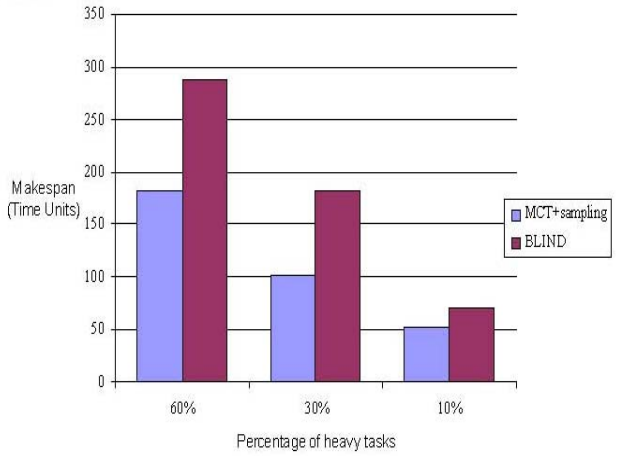

Figure3. Comparison of makespans observed for different percentages of expensive tasks, when either a blind heuristics or our MCT+sampling one is adopted.

In these first simulation tests, we essentially checked the feasibility of our approach. Our goal was thus to evaluate mapping quality, in terms of makespan, of an optimal on-line MCT+sampling technique. This mapper is optimal because it is supposed to also know in advance (through an oracle) the exact costs of the sampled tasks. In this way, we can evaluate the maximal improvement of our technique over the blind scheduling one.

Figures 2 illustrate two pairs of Gannt charts, which show the busy times of the six machines of our Grid testbed when tasks of different weights are submitted. In particular, each pair of charts is relative to two simulations, when either the blind or the MCT+sampling strategy is adopted. Machine i of cluster j is indicated with the label i[j]. Note that when the blind scheduling strategy is adopted, since cluster 0 is slower than the other and no datasets are moved, the makespan on the slower machines results higher. Note that our MCT+sampling strategy sensibly outperforms the blind one, although it introduces higher computational costs due to the sampling process. Finally, Figure 3 shows the improvements in makespans obtained by our technique over the blind one when the percentage of heavy tasks is varied.

4 Conclusion

In this paper we have discussed an on-line MCT heuristic strategy for scheduling high performance DM tasks onto a local organization of a Knowledge Grid. Scheduling decisions are taken on the basis of cost metrics and models based on information collected during previous executions, and use sampling to forecast execution costs. We have also reported the results of some preliminary simulations showing the improvements in the makespan (system throughput) of our strategy over a blind one. Our mapping and scheduling techniques might be adopted by a centralized on-line mapper, which is part of a more complex hierarchical Grid superscheduler, where the higher levels of the superscheduler might be responsible for taking rough schedule-decisions over multiple administrative organizations, e.g., by simply balancing the load among them by only considering aggregate queue lengths and computational power. The higher levels of a superscheduler, in fact, do not own the resources involved, may have outdated information about the load on these resources, and may be unable to exert any control over tasks currently on those domains.

The on-line mapper we have discussed does not permit node multitasking, and schedules tasks in batch. In future works we plan to consider also this feature, e.g., the mapper could choose to concurrently execute a compute-bound and an I/O-bound task on the same machine.

Finally, a possible drawback of our technique is the additional cost of sampling, even if it is worth considering that sampling has been already recognized as a feasible optimization technique in other fields, such as optimization of SQL queries. Of course, knowledge models extracted by sampling tasks could in some cases be of interest for the users, who might decide on the basis of the sampling results to abort or continue the execution on the whole dataset. On the other hand, since the results obtained with sampling actually represent a partial knowledge model extracted from a partition of the dataset, we could avoid to discard these partial results. For example, we might exploit a different DM algorithm, also suitable for distributed environments, where independent DM analysis are performed on different dataset partitions, and then the partial results are merged. According to this approach, the knowledge extracted from the sample D_i' might be retained, and subsequently merged with the one obtained by executing the task on the rest of the input dataset $D_i \backslash D_i$'.

References:

[1] GGF's working group on Grid scheduling dictionary,http://www.fz-juelich.de/zam/
[2] David Abramson, Rok Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations. In HPDC, pages 112–121, 1995.
[3] F. Berman and R. Wolski. The AppLeS Project: A Status Report, 1997.
[4] G. D. van Albada, J. Clinckemaillie, A. H. L. Emmen, J. Gehring, O. H Overeinder, A. Reinefeld, and P. M. A. Sloot. Dynamite - blasting obstacles to parallel cluster com-puting. In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and L. O. Hertzberger, editors, High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands, number 1593 in Lecture Notes in Computer Science, pages 300{310, Berlin, April 1999. Springer-Verlag.
[5] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. The Journal of Future Generation Computing Sys-tems, Jan 1999.
[6] Michael Litzkow, Miron Livny, and Matt Mutka, Condor-A Hunter of Idle Workstations. In Proc. The 8th International Conference of Distributed Computing Systems, San Jose, California, June, 1988, pp.204-111.
[7] James Frey, Todd Tannenbaum, et al, Condor-G: A Computation Management Agent for Multi-Institutional Grids. Journal of Cluster Computing, volume 5, pp. 237 - 246, 2002.
[8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. International Journal of Supercomputer Applications, 11(2):115-128, 1997.
[9] Globus Project website, http://wwww.globus.org
[10] F. Berman and R. Wolski. The AppLeS Project: A Status Report, 1997.
[11] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in legion. In 5th Workshop on Job Scheduling Strategies for Parallel Processing, in conjunction with the International Parallel and Distributed Processing Symposium, Apr 1999.
[12] M. J. Lewis and A. Grimshaw. The core legion object model. In Pternational Symposium on High Performance Distributed Computing. IEEE Computer Society Press, August 1996.
[13] Rajkumar Buyya, David Abramson, Jonathan Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), May 2000, Beijing, China. IEEE Computer Society Press, USA.
[14] D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Glocal Grid? In Proceedings of IPDPS2000.
[15] J. P. Bradford and J. Fortes. Performance and memory access characterization of data mining applications. In Proceedings of Workshop on Workload Characterization: Methodology and Case Studies, 1998.
[16] Francine Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao. Application level scheduling on distributed heterogeneous networks. In Proceedings of Supercomputing 1996, 1996.
[17] J. P. Bradford and J. Fortes. Performance and memory access characterization of data mining applications. In Proceedings of Workshop on Workload Characterization: Methodology and Case Studies, 1998.
[18] P. Becuzzi, M. Coppola, and M. Vanneschi. Mining of association rules in very large databases: a structured parallel approach. In Proc. of Europar, 1999.
[19] Kun Gao, Youquan Ji, Meiqun Liu, Jiaxun Chen: Rough Set Based Computation Times Estimation on Knowledge Grid. Lecture Notes in Computer Science 3470 Springer 2005, pages: 557-566
[20] Kun Gao, Kexiong Chen, Meiqun Liu, Jiaxun Chen: Rough Set Based Data Mining Tasks Scheduling on Knowledge Grid, Lecture Notes in Computer Science 3528 Springer 2005, Pages: 150-155
[21] Kun Gao: Predicting Grid Performance Based on Novel Reduct Algorithm, Lecture Notes in Computer Science 5178 Springer 2008, Pages: 289-296
[22] Kun Gao: A Uniform Parallel Optimization Method for Knowledge Discovery Grid, Lecture Notes in Computer Science 5178 Springer 2008, Pages: 306-312