Scheduling High Performance Data Mining Tasks on a Data Grid Environment

S. Orlando[1], P. Palmerini[1,2], R. Perego[2], F. Silvestri[2,3]

[1] Dipartimento di Informatica, Università Ca' Foscari, Venezia, Italy
[2] Istituto CNUCE, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy
[3] Dipartimento di Informatica, Università di Pisa, Italy

Abstract Increasingly the datasets used for data mining are becoming huge and physically distributed. 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 focus of this paper is on the core services of such a Grid infrastructure. In particular we concentrate our attention on the design and implementation of specialized Resource Allocation and Execution Management services aware of data source locations and resource needs of data mining tasks. Allocation and scheduling decisions are taken on the basis of performance cost metrics and models that exploit knowledge about previous executions, and use sampling to acquire estimate about execution behavior.

1 Introduction

In the last years we observed an explosive growth in the number and size of electronic data repositories. This gave researchers the opportunity to develop effective data mining (DM) techniques for discovering and extracting knowledge from huge amounts of information. Moreover, due to their size and also to social or legal restrictions that may prevent analysts from gathering data in a single site, the datasets are often physically distributed. If we also consider that data mining algorithms are computationally expensive, we can conclude that the *Grid* [5] is a natural platform for deploying a High Performance service for the Parallel and Distributed Knowledge Discovery (PDKD) process. The Grid environment may in fact furnish coordinated resource sharing, collaborative processing, and high performance data mining analysis of the huge amounts of data produced and stored. Since PDKD applications are typically *data intensive*, one of the main requirements of such a PDKD Grid environment is the efficient management of storage and communication resources.

A significative contribution in supporting data intensive applications is currently pursued within the Data Grid effort [2], where a data management architecture based on storage systems and metadata management services is provided. The data considered here are produced by several scientific laboratories geographically distributed among several institutions and countries. Data Grid

services are built on top of Globus [4], and simplify the task of managing computations that access distributed and large data sources. The above Data Grid framework seems to have a core of common requirements with the realization of a PDKD Grid, where data involved may originate from a larger variety of sources. Even if the Data Grid project is not explicitly concerned with data mining issues, its basic services could be exploited and extended to implement higher level grid services dealing with the process of discovering knowledge from larger and distributed data repositories. Motivated by these considerations, in [10] a specialized grid infrastructure named *Knowledge Grid* (K-Grid) has been proposed. This architecture was designed to be compatible with lower-level grid mechanisms and also with the Data Grid ones. The authors subdivide the K-Grid architecture into two layers: the core K-grid and the high level K-grid services. The former layer refers to services directly implemented on the top of generic grid services, the latter refers to services used to describe, develop and execute parallel and distributed knowledge discovery (PDKD) computations on the K-Grid. Moreover, the layer offers services to store and analyze the discovered knowledge.

In this paper we adopt the K-Grid architecture [10], and concentrate our attention on its core services, i.e. the Knowledge Directory Service (KDS) and the Resource Allocation and Execution Management (RAEM) services. The KDS extends the basic Globus Metacomputer Directory Service (MDS) [3], and is responsible for maintaining a description of all the data and tools used in the K-Grid. The metadata managed by the KDS are represented through XML documents stored in the Knowledge Metadata Repository (KMR). Metadata regard the following kind of objects: data sources characteristics, data management tools, data mining tools, mined data, and data visualization tools. Metadata representation for output mined data models may also adopt the *Predictive Model Markup Language* (PMML) [6] standard, which provides the XML specification for several kinds of data mining sources, models, and tools, also granting the interoperability among different PMML compliant tools.

The RAEM service provides a specialized *broker* of Grid resources for PDKD computations: given a user request for performing a DM analysis, the broker takes allocation and scheduling decisions, and builds the *execution plan*, establishing the sequence of actions that have to be performed in order to prepare execution (e.g., resource allocation, data and code deployment), actually execute the task, and return the results to the user. The execution plan has to satisfy given requirements (such as performance, response time, and mining algorithm) and constraints (such as data locations, available computing power, storage size, memory, network bandwidth and latency). Once the execution plan is built, it is passed to the Grid Resource Management service for execution. Clearly, many different execution plans can be devised, and the RAEM service has to choose the one which maximizes or minimizes some metrics of interest (e.g. throughput, average service time).

In this paper we analyze some of the issues encountered in the design and implementation of an allocation and scheduling strategy for the RAEM service,

i.e. for the broker of the K-Grid architecture. In its decision making process, this service has to exploit a composite performance model which consider the actual status of the Grid, the location of data sources, and the task execution behavior. The broker needs quite detailed knowledge about computation and communication costs to evaluate the profitability of alternative mappings and related dataset transfer/partitioning. For example, the broker could evaluate when it is profitable to launch a given expensive mining analysis in parallel. Unfortunately, the performance costs of many DM tools depend not only on the size of data, but also on the specific mining parameters provided by the user. Consider for example the Association Rule Mining (ARM) analysis: its complexity not only depends on the size of the input dataset, but also on the user-provided support and confidence thresholds. Moreover, the correlations between the items present in the various transactions of a dataset largely influences the number and the maximal length of the rules found by an ARM tool. Therefore, it becomes difficult to predict in advance either the computational and input/output costs, or the size of the output data.

In order to deal with these issues, we propose to include in the KDS service *dynamic* information about the performances of the various DM tools over specific data sources. This information can be added as additional metadata associated with datasets, and collected by monitoring previous runs of the various software components on the specific datasets. Unfortunately these metadata may not be available when, for example, a dataset is analyzed for the first time. In the absence of knowledge about costs, the Grid RAEM service would make blind allocation and scheduling decisions. To overcome this problem, we suggest to exploit *sampling* as a method to acquire preventive knowledge about the rough execution costs of specific, possibly expensive, DM jobs. Sampling has also been suggested as an efficient and effective approach to speed up data mining process, since in some case it may be possible to extract accurate knowledge from a sample of an huge dataset [11]. Unfortunately, the accuracy of the mined knowledge depends on the size of the sample in a non-linear way, and determining how much data has to be used is not possible a priori, thus making the approach impractical. In this paper we investigate an alternative use of sampling: in order to forecast the actual execution cost of a given DM algorithm on the whole dataset, we run the same algorithm on a small sample of the dataset. Many DM algorithms demonstrate optimal scalability with respect to the size of the processed dataset, thus making our performance estimate possible and accurate enough. Moreover, even if a wrong estimate is made, this can only affect the optimal use of the Grid and not the results of the final DM analysis to be performed on the whole dataset. Besides execution costs, with sampling we can also estimate the size of the mined results, as well as to predict the amount of I/O and main memory required. These costs will thus feed specific performance models used by the K-Grid scheduler in order to forecast communication overhead, the effect of resource sharing, and the possible gain deriving from parallelism exploitation.

The paper is organized as follows. Section 2 introduce our K-Grid scheduler and presents the cost model on which it is based on. In Section 3 we discuss the

methodology to be used to predict performances by sampling. Section 4 discusses our mapper and the relative simulative framework, and reports some preliminary results. Finally, Section 5 draws some conclusions and outlines future work.

2 Distributed Scheduling on the Knowledge Grid

In general a Grid broker should perform the following actions: (1) discover a number of resources that fit with the minimum requirements for the execution; (2) verify permissions for submitting a job on these resources; (3) select the best matching resources with respect to the application performance requirements, and schedule the job. Existing broker for Grids fits more or less with the model above. The main difference regard their organization (e.g., centralized, hierarchical, distributed), and the scheduling policy (e.g., we may optimize system throughput or application completion time). Moreover, scheduling algorithms may consider the state of the system as unchanged in the future, i.e. only depending on the decisions taken by the scheduler, or may try to *predict* possible changes in the system state. In both cases, it is important to know in advance information about task durations under several resource availability constraints and possible resource sharing. The algorithms used to schedule jobs may be classified as dynamic or static [9]. Dynamic scheduling may be on-line, i.e. when a task is assigned to a machine as soon as it arrives, or batch, i.e. when the arrived tasks are collected into a set that is examined for mapping at pre-scheduled times. On the other hand, static approaches, which exploit very expensive mapping strategies, are usually adopted to map long-running applications. Due to the characteristics of DM jobs, which are often interactive, we believe that the best scheduling policy to be used in the design of a K-Grid scheduler should be a dynamic one. In this preliminary study, we thus evaluate feasibility and benefits of adopting a *centralized local on-line scheduler* for a Grid organization, which includes several clusters connected to the Grid, and may be shared by several Virtual Organizations (VO). This local scheduler will be part of a more complex hierarchical superscheduler for our K-Grid. The only performance measure considered in this work is completion time of DM jobs in order to optimize system throughput, while the constraints regard data access, computing power, memory size, and network bandwidth.

2.1 Task scheduling issues

Before sketching the dynamic scheduling algorithm used for mapping DM jobs, we introduce the issues encountered in scheduling these kind of computations and the simple cost model proposed. Most of the terminology used and the model of performance adopted have been inspired by [7,9].

Several decisions have to be taken in order to map and schedule the execution of a given (data mining) task on the Grid. First consider that a DM task t_i is completely defined in terms of the DM analysis requested, the dataset $\mathcal{D}_i$ (of size $|\mathcal{D}_i|$) to analyze, and the user parameters u_i that specify and affect the analysis

behavior. Let $\alpha_i(\mathcal{D}_i)$ be the knowledge model extracted by t_i, where $|\alpha_i(\mathcal{D}_i)|$ is its size. In general the knowledge model extracted has to to be returned to a given site, where further analysis or visualization must be performed. Before discussing in detail the mapping algorithm and the simulation environment, let us make the following assumptions:

- A centralized local scheduler controls the mapping of DM tasks onto a small Grid organization, which is composed of a set $\mathcal{M} = \{m_1, \ldots, m_{|\mathcal{M}|}\}$ of $|\mathcal{M}|$ machines, where a known performance factor p_j is associated with each machine m_j. This performance factor measures the relative speed of the various machines. Since in this paper we do not consider node multitasking, we do not take into account possible external machine loads that could affect these performance factors. Moreover, the machines are organized as a set of clusters $\mathcal{CL} = \{cl_1, \ldots, cl_{|\mathcal{CL}|}\}$, where each cluster cl_J comprises a disjoint set of machines in $\mathcal{M}$ interconnected by a high-speed network. In particular, $cl_J = \{m^J_1, \ldots, m^J_{|cl_J|}\}$. Each cluster cl_J is thus a candidate for hosting a parallel implementation of a given DM analysis. The performance factors of a cluster cl_J is p_J, which is equal to the factor of the slowest machine in the cluster.
- The code (sequential or parallel) that implements each DM tool is considered to be available at each Grid site. So the mapping issues, i.e. the evaluation of the benefits deriving from the assignment of a task to a given machine, only concern the communication times needed to move input/output data, and also the *ready times* of machines and communication links.
- On the basis of sampling or historical data we assume that it is possible to estimate e_i, defined as the base (normalized) sequential computational cost of task t_i, when executed on dataset $\mathcal{D}_i$ with user parameters u_i. Let $e_{ij} = p_j \cdot e_i$ be the execution time of t_i on machine m_j.
  When an analysis is performed in parallel on a cluster cl_J, we assume that, in the absence of load imbalance, task t_i can be executed in parallel with a quasi perfect speedup. In particular, let e_{iJ} be the execution time of task t_i on a cluster cl_J, defined as $max_{m^J_t \in cl_J}(e_{it}/|cl_J|) + ovh = max_{m^J_t \in cl_J}((p_t \cdot e_i)/|cl_J|) + ovh$. The term ovh models the overhead due to the parallelization and heterogeneity of the cluster. Consider that when a cluster is homogeneous and e_i is large enough, ovh is usually very small.
- A dataset $\mathcal{D}_i$ may be centralized, i.e. stored in a single site, or distributed. In the following we will not consider the inherent distribution of datasets, even if we could easily add such a constraint into our framework. So we only assume that a dataset is moved when it is advantageous for reducing the completion time of a job. In particular, a centralized dataset stored in site h can be moved to another site j, and the cost depends on the average network bandwidth b_{hj} between the two sites. For example, $\mathcal{D}_i$ can be transferred with a cost of $\frac{|\mathcal{D}_i|}{b_{hj}}$.
  Moving datasets between sites has to be carried out by the *replica manager* of the lower Grid services, which is also responsible of the coherence of copies. Future accesses to the a dataset may take advantage of the existence of

different copies disseminated on the Grid. So, when a task t_i must be mapped, we have to consider that, for each machine, we have to choose the most advantageous copy of a dataset to be moved or accessed.

2.2 Cost model

In the following cost model we assume that each input dataset is initially stored on at least 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 replicated onto other machines, or partitioned among the machines composing a cluster.

Sequential execution. Dataset $\mathcal{D}_i$ 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 $|\alpha_i(\mathcal{D}_i)|$ must be returned to machine m_k. We have to consider the communications needed to move $\mathcal{D}_i$ 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:

$$E_{ij} = \frac{|\mathcal{D}_i|}{b_{hj}} \; + \; e_{ij} \; + \; \frac{|\alpha_i(\mathcal{D}_i|)}{b_{jk}}$$

Parallel execution. Task t_i is executed in parallel on a cluster cl_J, with an execution time of e_{iJ}. In general, we have also to consider the communications needed to move and partition $\mathcal{D}_i$ from machine m_h to cluster cl_J, and to return the results $|\alpha_i(\mathcal{D}_i)|$ 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 cl_J. The total execution time is thus:

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

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.

Performance metrics. To optimize scheduling, our batch 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 $\mathcal{D}_i$ from m_h to m_j, s_j the starting

time of the sequential execution of task t_i on m_j, and, finally, s_{jk} the starting time of the communication needed to move $\alpha_i(\mathcal{D}_i)$ from m_j to m_k. From the above definitions:

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

where $\delta_1 \ = \ s_j - (s_{hj} + \frac{|\mathcal{D}_i|}{b_{hj}}) \ \geq \ 0$ and $\delta_2 = s_{jk} - (s_j + e_{ij}) \ \geq 0$.

If $m_{\bar{j}}$ 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 $C_i = C_{i\bar{j}}$. 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.

3 Sampling as a method for performance prediction

Before discussing our mapping strategy based on the cost model outlined in the previous Section, we want to discuss the feasibility of *sampling* as a method to predict the performance of a given DM analysis. 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 [11], 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.

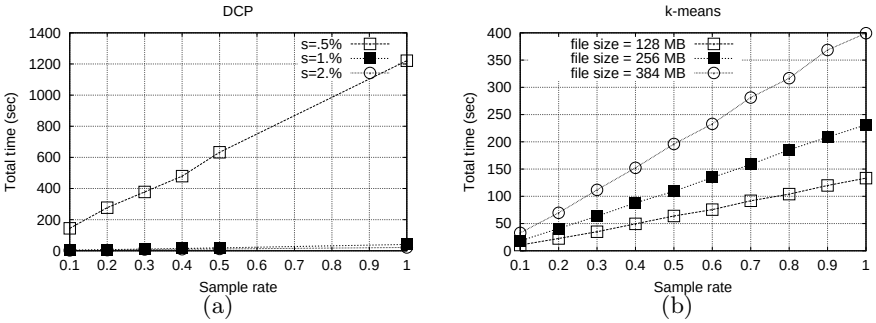

Figure1. Execution time of the DCP ARM algorithm (a), and the k-means clustering one (b), as a function of the sample rate of the input dataset.

Suppose that a given task t_i is first executed on a sample $\widehat{\mathcal{D}}_i$ of dataset $\mathcal{D}_i$ on machine m_j. Let $\widehat{e}_{ij}$ be this execution time, and let $\widehat{e}_i \ = \ \widehat{e}_{ij}/p_j$ 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 $e_i = F(|\mathcal{D}_i|)$. In particular, the coefficients of $F()$ must be derived on the basis of the sampled execution, i.e., in terms of $\widehat{e}_i$, $\widehat{\mathcal{D}}_i$, and $|\widehat{\mathcal{D}}_i|$. The simplest case is when the algorithm scales linearly, so that $F()$ is a linear function of the size of the dataset, i.e. $e_i = \gamma \cdot |\mathcal{D}_i|$, where $\gamma = \widehat{e}_i \ / \ |\widehat{\mathcal{D}}_i|$.

We analyzed two DM algorithms: DCP, an ARM algorithm which exploits out-of-core techniques to enhance scalability [8], and *k-means* [1], 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 $\mathcal{D}_i$. 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 $\widehat{\mathcal{D}}_i$. 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-core 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 [11], 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 [7,9], 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*

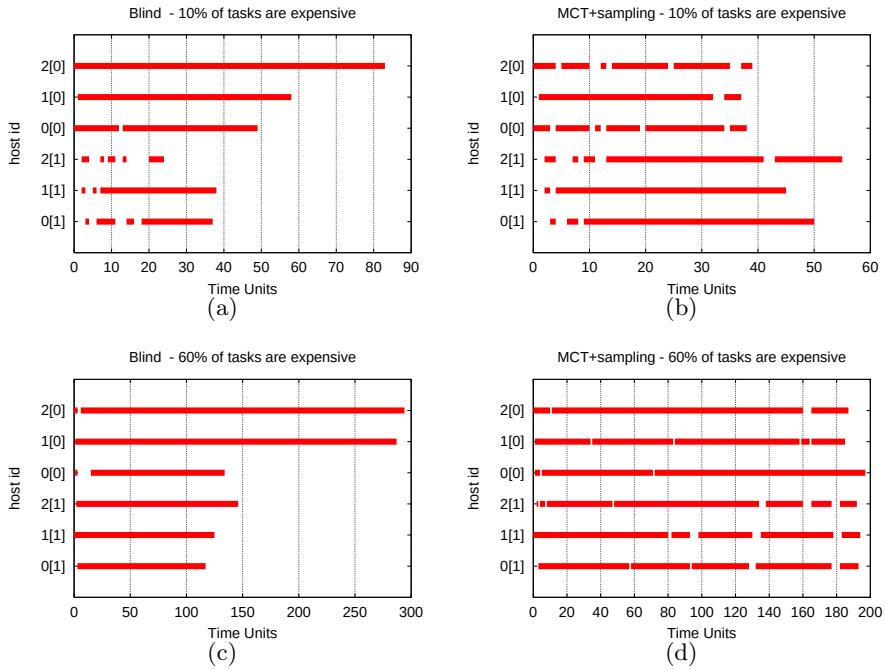

Figure2. Gannt charts showing the busy times (in time units of 100 sec.) of our six machines when either the 10% (a,b) or the 60% (c,d) of the tasks are expensive: (a,b) blind scheduling heuristics, (c,d) MCT+sampling scheduling heuristics.

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 $\widehat{t_i}$, i.e. the task t_i executed on the sampled dataset $\widehat{\mathcal{D}_i}$. Unfortunately, the expected execution time of sampled task $\widehat{t_i}$ 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 $\widehat{t_i}$, it simply assigns $\widehat{t_i}$ to the machine that hosts the corresponding input dataset, so that no data transfers are involved in the execution of $\widehat{t_i}$. When $\widehat{t_i}$ 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.

4.1 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 b_{WAN} (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 D_i are all of medium size (50MB), and are randomly located on the machines belonging to the two clusters.

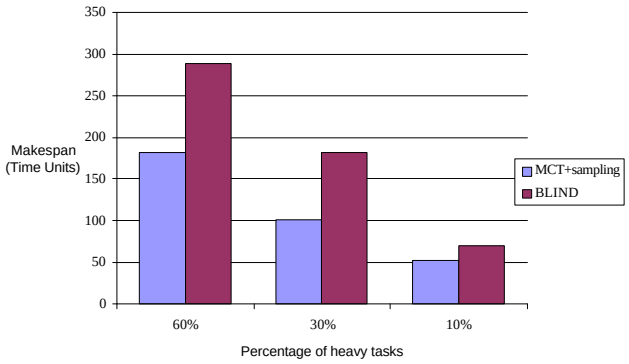

Figure3. Comparison of makespans observed for different percentages of expensive tasks, when either a blind heuristics or our MCT+sampling one is adopted.

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.

5 Conclusions and Future Works

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 $\widehat{\mathcal{D}}_i$ might be retained, and subsequently merged with the one obtained by executing the task on the rest of the input dataset $\mathcal{D}_i \setminus \widehat{\mathcal{D}}_i$.

References

1. R. Baraglia, D. Laforenza, S. Orlando, P. Palmerini, and R. Perego. Implementation issues in the design of I/O intensive data mining applications on clusters of workstations. In *Proc. of the 3rd Workshop on High Performance Data Mining, Cancun, Mexico*. Spinger-Verlag, 2000.
2. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: towards an architecture for the distributed management and analysis of large scientific datasets. *J. of Network and Comp. Appl.*, (23):187–200, 2001.
3. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press, 1997.
4. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. of Supercomputer Applications*, 11(2):115–128, 1997.
5. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. 1999.
6. The Data Mining Group. PMML 2.0. `http://www.dmg.org/pmmlspecs_v2/pmml_v2_0.html`.
7. M. Maheswaran, A. Shoukat, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *8th HCW*, 1999.
8. S. Orlando, P. Palmerini, and R. Perego. Enhancing the Apriori Algorithm for Frequent Set Counting. In *Proc. of 3rd Int. Conf. DaWaK 01 - Munich, Germany*. LNCS Spinger-Verlag, 2001.
9. H. J. Siegel and Shoukat Ali. Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems. *Journal of Systems Architecture*, (46):627–639, 2000.
10. D. Talia and M. Cannataro. Knowledge grid: An architecture for distributed knowledge discovery. *Comm. of the ACM*, 2002. to appear.
11. M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. In *7th Int. Work. on Research Issues in Data Eng.*, pages 42–50, 1997.