A Sampling-based Scheduling Method for Distributed Computing

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. 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. We adopt the sampling method for process the tasks scheduling in distributed data mining environment. The experimental results show the sampling method is applicable to task scheduling in dynamic environment and can be adopted to obtain a higher result.

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

1 Introduction
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 resources 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], maximizing 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 assumptions and constraints. In the following section, I attempt to outline the features of each system and summarize 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.

3.2 Condor
Condor [4] 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 checkpointing 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 workstation 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 checkpointing, 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 system-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 resources 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 [5], leverages the advantages of both condor and Globus Toolkit [6] [7]. Globus Toolkit is a software infrastructure for setting up a Grid environment across multiple administrative domains, which supports 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 workstations 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 Security Infrastructure) 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 resources. 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.

In the new applications of Grid computing, some significative efforts were present in [12][13][14][15]. 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 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 addition 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 [8], 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 $\hat{D}_i$ of dataset $D_i$ on machine $m_j$. Let $\hat{e}_{ij}$ be this execution time, and let $\hat{e}_i = \hat{e}_{ij} / p_j$ be the normalized execution time on the sample. Sampling is feasible as a method to predict performance of task $t_i$, if, 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 $\hat{e}_i$, $\hat{D}_i$, and $|\hat{D}_i|$. 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 $\gamma = \hat{e}_i / |\hat{D}_i|$. 

We analyzed two DM algorithms: DCP, an ARM algorithm which exploits out-of-core techniques to enhance scalability [9], 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 \( \hat{D}_i \). 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 \([10]\), 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 \([11]\), 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 \( \hat{t}_i \), i.e. the task \( t_i \) executed on the sampled dataset \( \hat{D}_i \). Unfortunately, the expected execution time of sampled task \( \hat{t}_i \) 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 \( \hat{t}_i \), it simply assigns \( \hat{t}_i \) to the machine that hosts the corresponding input dataset, so that no data transfers are involved in the execution of \( \hat{t}_i \). When \( \hat{t}_i \) 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 \( b_{\text{WAN}} \) and \( b_{\text{LAN}} \) 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 \( b_{\text{WAN}} \) (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.

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 Gantt 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 
\( D_i \setminus \hat{D}_i \).

References: