Meta-Scheduling in Advance using Red-Black Trees in Heterogeneous Grids

Luis Tomás, Carmen Carrión, Blanca Caminero
Dept. of Computing Systems
The University of Castilla La Mancha.
Albacete, Spain
{luisbt, carmen, blanca}@dsi.uclm.es

Agustín Caminero
Dept. of Communication and Control Systems
The National University of Distance Education.
Madrid, Spain
accaminero@scc.uned.es

Abstract—The provision of Quality of Service in Grid environments is still an open issue that needs attention from the research community. One way of contributing to the provision QoS in Grids is by performing meta-scheduling of jobs in advance, that is, jobs are scheduled some time before they are actually executed. In this way, the appropriate resources will be available to run the job when needed, so that QoS requirements (i.e., deadline) are met.

This paper presents two new techniques, implemented over the red-black tree data structure, to manage the idle/busy periods of resources. One of them takes into account the heterogeneity of resources when estimating the execution times of jobs. A performance evaluation using a real testbed is presented that illustrates the efficiency of this approach to meet the QoS requirements of users.

I. INTRODUCTION

Transparent access to resources of very different nature, like CPU, network, data or software, should be provided by the Grid. The Grid resources are in different domains under different access policies, so searching and using these resources is a hard task for Grid users. Also, carrying out this process manually is not feasible in a large-scale Grid environment with many potentially available resources. Therefore, the Grid infrastructure must provide the needed services for automatic resource brokerage that take care of the resource selection and negotiation process [1]. This infrastructure is named “meta-scheduler” and hides this process from the user.

There are core Grid services which provide the basic functionality for setting up Grids. One of them is the Globus Toolkit [2], which is the current de-facto standard for many Grid projects. It has interfaces for accessing remote resources, security services and information services among others. However, higher-level services for coordinating the resource access are still missing in this toolkit.

The user’s experience of the Grid is determined by the functionality and performance of the resource broker and job submission components. But the heterogeneous and distributed nature of the Grid as well as the differing characteristics of different applications complicate the brokering problem. To further complicate matters, the broker typically lacks total control and even complete knowledge of the state of the resources [3].

Current scheduling systems adopt three different approaches to tackle these problems [4]: scheduling based on just-in-time information from Grid Information Service (GIS), performance prediction, and dynamic rescheduling at run time. Getting resource static information, such as CPU frequency, memory size, network bandwidth or file system is feasible. But run-time information, such as CPU load, available memory, and available network bandwidth, is more difficult to obtain. This is because of performance fluctuation which in turn is due to contention among shared resources.

The key idea to solve this problem is to ensure that a specific resource is available when the job requires it. So, it is necessary to reserve or schedule the use of resources in-advance [5]. Advanced reservation can be defined as a restrictive or limited delegation of particular resource capacity. This delegation is for a defined time interval and is obtained by a request through a negotiation process [6]. The objective of such advanced reservation is to provide some QoS by ensuring that a certain job ends on time.

In the Grid, advanced reservation has been largely ignored due to the dynamic Grid behaviour, underutilization concerns, multi-constrained applications, and lack of support for agreement enforcement. These issues force the Grid middleware to make resource allocations at runtime with reduced QoS. However, advanced reservation mechanisms enable QoS agreements with users and increase the predictability of a Grid system. Notice that prediction plays quite an important role in the decision-making phase [7].

On the other hand, incorporating such mechanisms into current Grid environments has proven to be a challenging task due to the resulting resource fragmentation [8]. Also, the main challenge of advanced reservations is clear: without knowing the exact status of the resources at future points in time it is difficult to decide whether a job with a certain requirements can be executed fulfilling this QoS [9].

This paper proposes a new scheduling algorithm to tackle the scheduling in advance problem. This algorithm is concerned with the dynamic behaviour of the Grid resources, the use of them, and the characteristics of the jobs.

Authors use red-black trees to manage the usage of resources. This idea has already been tried in [8], [9], where authors assume that users have a priori knowledge on the
duration of jobs – which may not be true most of times. In the present work, estimations for the duration of jobs are calculated based on (1) a linear function and (2) executions log data. In the first case, the resources are treated as homogeneous (the same function is used to calculate the duration of jobs in all the resources), whilst in the second case, estimations on job durations take into account the resource where previous executions took place, thus paying attention to the heterogeneity of Grid resources. These two ways of estimating the execution time of jobs are presented and evaluated in this paper.

The paper is organized as follow. Related work is presented in Section II. In Section III a brief overview of the general meta-scheduling framework is presented. Section IV explains the implemented extensions. Section V presents the experiments carried out for evaluating the extensions. Finally, conclusions obtained and suggested guidelines for future work are included in Section VI.

II. RELATED WORK

A Grid application may need multiple heterogeneous resources which may span over administrative boundaries, thus making the management of resources a challenging task [10].

Software infrastructures required for resource management and other tasks such as security, information dissemination and remote access are provided through Grid toolkits such as Globus [2] and Legion [11].

Regarding advanced reservations of resources, Globus Architecture for Reservation and Allocation (GARA) [12] was introduced later for application-level dynamic scheduling of collection of resources, co-allocation and advanced reservations. GARA is one of the seminal works on advanced reservation and defines a basic architecture and simple API for the manipulation of advanced reservation of different resources.

Since then, advanced reservations have been studied in numerous contexts. For example, several systems have been developed, like Maui Scheduler [13], that allow advanced scheduling with advanced reservations for cluster environments.

Among the systems that allow resource reservation in a Grid we can find Grid Capacity Planning [7], that provides users with reservations of Grid resources through negotiations, co-allocations and pricing.

Overall, the great interest that Grid community pays to advanced reservation of resources is because it provides planning capabilities to user applications. Despite support for reservation in the underlying infrastructure is currently limited, advanced reservation feature is required to meet QoS guarantees in Grid environments, as several contributions conclude [7], [12]. Qu [14] describes a method to overcome this shortcoming by adding a Grid advanced reservation manager on top of the local scheduler(s). Advanced reservations can hence be provided regardless of whether the local scheduler supports them. This reservation approach however requires that all job requests are passed through the Grid advanced reservation manager. The performance penalty imposed by the usage of advanced reservations (typically decreased resource utilization) has been studied in [15].

Furthermore, advanced reservations have been shown to increase the predictability of the system while maximizing its flexibility and its adaptability to cope with the dynamic behaviour of grid environments [16]. In [7] a cost-aware resource model is presented in which reservation for each application task is performed separately by negotiating with the resource provider. They model resource allocation as an on-line strip packing problem and introduce a new mechanism that optimizes resource utilization and QoS constraints while generation the contention-free solutions.

This work uses red-black trees to manage the usage of resources, and it has already been tried in [8], [9], where authors assume that users have a priori knowledge on the duration of jobs. In [8], a technique to perform scheduling using homogeneous resources is presented, whilst [9] presents a technique to perform scheduling using heterogeneous resources. In the present work, we do not consider such a priori knowledge, so we calculate estimations on the execution time of jobs. This paper presents two ways of calculating such estimations (namely linear function and log-based predictions), which are explained in Section IV-C.

III. META-SCHEDULING IN ADVANCE

A Grid is an environment in which resources vary dynamically – they may fail, join or leave the Grid system at any time. Also, such dynamity comes from the fact that every Grid resource needs to execute local tasks as well as tasks from Grid applications.

From the viewpoint of a Grid application all the tasks from both local users and Grid users are loads on the resource. So, everything in the system is evaluated by its influence on the application execution. The Grid meta-scheduler may obtain such load information from the GIS or a resource monitor, and decide where to execute Grid tasks based on such information.

Support for advanced reservations of resources plays a key role in Grid resource management as it allows the system to meet user expectations with regard to time requirements and temporal dependence of applications, and increases the predictability of the system [9].

A Grid advanced reservation process can be divided into two steps [6]:

1. Meta-scheduling in advance: Select the resources to execute the job, and the time period when the execution will be performed. There are two concepts: requesting a reservation and committing a reservation. A reservation
request contains the start time and the requested length of the reservation. For committing a reservation, the metascheduler uploads a commit message containing the job id. At this moment, the job starts its execution in the previously selected computing resource.

2. Negotiation for resource reservation: Consists of the physical reservation of the resources needed for the job, which may not always be possible.

There is a great scepticism in the Grid community about advanced reservations ability in spite of their attractive features; this fact is mainly due to three reasons [9]. First, advanced reservations cause severe performance degradation [15] because algorithms are very complex. Second, typical advanced reservation mechanisms lack flexibility as they do not permit graceful degradation in application performance [17]. They are made in advance and it is not possible to know the exact status of all the Grid resources in the future. And third, existing approaches suffer from poor scalability as they are not effective in managing large sets of advanced reservations or handling resource fragmentation.

To overcome these challenges, algorithms for advanced reservations need to be efficient so they can adapt to dynamic changes in resource availability and user demand without hurting system and user performance. Moreover, they must take into account resource heterogeneity since resources in Grid environments are typically highly heterogeneous.

Scheduling advanced reservation requests with a given start time, execution time and a given amount of laxity is an NP-Hard problem [10]. For this reason, it could be useful to employ techniques from computational geometry to develop an efficient heterogeneity-aware scheduling algorithm [9].

The start time defines the sooner time jobs can start to be executed, and the deadline the time by which the job must have been executed. Also, the deadline is a measure of the QoS required by the user [8]. If it is determined that the deadline cannot be met, the job is dropped and its user is notified accordingly. The execution time is the estimated duration of the job. Finally, laxity represents how flexible the user is in the execution of his job.

In a real Grid environment, reservations may not be always feasible, since not all the Local Resource Management Systems (LRMS) permit them. Apart from that, if there are other types of resources such as bandwidth (e.g. the Internet), which lack any management entity, and makes impossible their reservation. This is the reason to perform meta-scheduling in advance rather than advanced reservations to provide QoS in Grids.

This work proposes a framework in which the meta-scheduler does not perform a physical reservation, but it keeps track of the meta-scheduling decisions already made in order to make future decisions. Thus, if only Grid load exist, this would be enough to provide QoS since the meta-scheduler would not overlap jobs on resources.

In this work we focus on the type of applications where the user provides both the input files and the application itself. Also, these jobs – called simple jobs– do not have workflow dependencies. However, with the start time and the deadline of each job, we can set a specific workflow among jobs. Taking into account these assumptions an scheduling in advance process is done following these steps:

1) First, a “user request” specifying the job QoS requirements is received. For example, “start time” or “deadline”. In this work, these requirements are based on finishing its execution before a deadline.

2) The meta-scheduler executes a gap search algorithm. This algorithm takes into account the decisions already made, the status reported by the resources and the QoS requirements of the job to obtain the resource and the time interval to be assigned for the execution of this job.

3) If it is not possible to fulfill the QoS requirements using the resources of its own domain, the communication with meta-schedulers allocated in other domains starts.

4) If it is still not possible to meet the QoS requirements, a negotiation process with the user is started in order to define new QoS requirements.

A good running time prediction of tasks is very helpful and important for job scheduling and resource management in Grids. Such prediction information can be derived using information about the application, such as the running time of previous similar tasks [18].

Since the execution time of a computation-intensive task on a host is tightly related to the CPU load in that host while the job is being run, we can make use of the information about the CPU load to predict the task running time. Thus, if it is possible to predict the load on a host during the execution of a task, we could predict the execution time of the task on that host. However, in a Grid environment, resource contention causes host load and availability to vary over time, and makes the load prediction problem more difficult [18]. Therefore, it is quite difficult for the Grid meta-scheduler to estimate the exact cost of a task execution on different sites [4]. Moreover, independence and autonomy of domains is another obstacle, since domains may not want to share information on the load of their resources with other domains, thus making predictions unfeasible.

IV. IMPLEMENTATION OF META-SCHEDULING IN ADVANCE

This section depicts the implementation performed allowing meta-scheduling in advance. First, the policies for selecting and allocating the jobs into resources gap are explained. Then, the data structure used for managing this information is shown. Finally, the prediction needs are commented.

It must be noted that all the proposals are going to be evaluated in a real environment. This implementation
job mains. When using GridWay, users need to generate a
organization or scattered across several administrative do-
ferent local resource management systems, within a single
efficient sharing of computing resources, managed by dif-
scheduler [19]. GridWay enables large-scale, reliable and
has been performed as an extension to the GridWay meta-
nerable and efficient sharing of computing resources, managed by dif-
different local resource management systems, within a single
an efficient sharing of computing resources, managed by dif-
resource discovery and monitoring, job submission and
enables the natural implementation of a variety of strategies
A single tree structure is appropriate for immediate dead-
plane as Figure 2 [8], [9] depicts. The job coordinates are
of resource discovery and monitoring, job submission and
In this implementation, resource usages are divided into
time slots of 1 minute. This is a customizable parameter.
Then, we have to schedule the future usage of these slots
by assigning the jobs into resources at one specific time
(taking one or more time slots). For this reason, allocation
policies to find the best slots for each job and data structures
to keep a trace of the usage of slots are needed. These are
explained the next.

A. Allocation policies

It is very important how to allocate the jobs into time
slots since this allocation influences how many jobs can
be scheduled because of generated fragmentation. Different
ways of searching and allocating jobs into resources can
be developed considering both the already scheduled jobs
and the generated fragmentation. In this work, fragmentation
refers to the free time slots in between two allocations.

In our first approach, a First Fit policy has been consid-
ered. This technique selects the first free gap found to ac-
commodate the new job. It can present a big fragmentation,
as a result of which many jobs may be rejected. There also
exist other techniques like Best Fit. This policy selects the
free gap which better fits with the number of slots required
by the job. Fragments created are smaller because of that,
thus it is harder to use those free slots to allocate new jobs.

B. Data structure

The data structure is a key aspect since improvements can
be achieved in the algorithms depending on the type of data
structures that are used for processing the available gaps.
A suitable data structure yields better execution times and
also reduces the complexity of algorithms. Furthermore, the
data structure will also have influence on how scalable the
algorithm can be.

This work uses red black trees [8], [9]. The objective of
using red-black trees is to develop techniques to efficiently
identify feasible idle periods for each arriving job request,
without having to examine all idle periods.

A single tree structure is appropriate for immediate dead-
lines, in which each job is represented by a single point in
the plane as Figure 2 [8], [9] depicts. The job coordinates are
job starting time and job ending time. Besides, partitioning
the idle periods into subsets (strips, 2l_min size in the figure)
enables the natural implementation of a variety of strategies
for selecting one among multiple feasible idle periods.

As Castillo et al. explain in [8], the trees can be divided
into two sections, named R1 and R2, as Figure 2 [8], [9]
depicts. R1 region represents the gaps which start at or
before the job’s ready time. Therefore, any idle period in
this region can accommodate the new job without delaying
its execution. R2 region represents the gaps which start later
than the job’s ready time but these are large enough for it.

In this figure, labeled points represent the idle periods
(gaps) with its start and finish time. P represents the earliest
start and ending times, whilst P’ represents the latest, for the
current job. Thus, the line between P and P’ represents when
a new job can be scheduled. All the points above and to the
right of this line (regions R1 and R2) represent possible gaps
to allocate the job.

A job scheduled in an idle period will create at most two
new idle periods: one between the beginning of the gap and
the start of the job (the leading idle period), and one after
the end of the job and the end of the original idle period (the
trailing idle period). The leading idle period will have zero
length at any point in the region R2, since the start time of
this gap is later than the job start time. Note also that the later
the starting time of a gap the longer the execution of the new

Figure 1. The Scheduler in Advance Layer (SA-layer).
job will be delayed. This suggests that the strips of region R2 should be searched from top to bottom to minimize the job turnaround time.

C. Job execution time estimation

The performance differences between Grid resources and the fact that their relative performance characteristics may vary for different applications, makes predictions of job execution time difficult. Techniques for such predictions include applying statistical models to previous executions and heuristics based on job and resource characteristics [20][21][22].

Based on this, the algorithm proposed by Claris et al. [8],[9] is extended in order to take into account the heterogeneity of Grid resources. This work compares the usage of historical data logs and a linear function to calculate the job execution time. The monitoring information collected is kept in a database and reused for the next resource allocation decisions.

The new implemented SA-layer also stores information concerning previous application executions (called DB Executions in Figure 1), and the status of resources and network over time (called DB Resources in Figure 1). By processing these information logs, a more accurate estimation of the execution time of the job in the different computational resources can be performed. Recall that the memory overhead is negligible (about several Mbits). Even these data logs can be splitted and distributed.

As mentioned before, two ways of calculating estimations for job execution times are presented in this paper: based on a linear function, and based on executions log data. The linear function does not take into account the different resource performance, only the input parameters of the job and the knowledge about its behaviour. So that, using this kind of estimation to predict the job execution time, all the resources are treated as homogeneous.

On the other hand, using data logs, the resource heterogeneity is taken into account. The mean of the execution time from previous executions for each type of application is calculated. Two applications are considered to belong to the same type when they have the same input and output parameters – in terms of number, type and size. This mean is calculated for each host separately, taking into account the host where previous executions were performed. Then, the mean is used as a prediction for the execution time in next executions of each type of application for each host. This way, predictions on the overall completion time are calculated for each type of application for each host in the system. These predictions are only calculated when a suitable gap has been found in the host, so that there is no need to calculate the completion times for all the hosts in the system – which would be quite inefficient.

V. Experiments and results

This section describes the experiments conducted to test the usefulness of this work, along with the results obtained.

A. Testbed description

The evaluation of the scheduling in advance implementations has been carried out in a local real Grid environment. Figure 3 shows the topology used as testbed. The testbed is composed by resources which are in two different buildings. In one building, named Instituto de Investigación en Informática de Albacete (I3A), there are, on the one hand, the GridWayI3A.uclm.es machine that carries out the scheduler tasks, and on the other hand, several computational resources named GlobusX.uclm.es with X [1, 10]. In the other building, named Escuela Superior de Ingeniería Informática (ESII), there is a cluster machine with 88 cores. As it can be seen at Figure 3, all these machines belong to the same administrative domain (UCLM) but they are located in different subnets. Notice that these machines belong to other users, so they have their own local background load.
B. Workload used

One of the GRASP [23] benchmarks, named 3node, has been run to evaluate the implementation. The 3node test consists of sending a file from a “source node” to a “computation node”, which performs a search pattern, generating an output file with the number of successes. This test is meant to mimic a pipelined application that obtains data at one site, computes a result on that data at another, and analyses the result on a third site. Furthermore, this test has parameterizable options to make it more computing intensive (compute_scale parameter) and / or more network demanding (output_scale parameter).

There are important parameters that have to be taken into account in the workload used for measuring performance, as can be seen in Figure 4. “T_max reservation” represents the advance with which we can make an scheduling in advance; “TExec_i” is the time needed to complete the job i; “Scheduling Window” shows the time interval in which the job has to be scheduled; “Arrival Ratio” depicts the average time between two jobs sent; and “Laxity” represent how strict the user is when scheduling a job, which is the difference between the “Scheduling Window” and the “TExec_i” for a job.

For this evaluation, both the compute_scale and the output_scale take values between 0 and 20, being the average 10. The T_max reservation is set to 0 because the SA-layer (with both prediction techniques) is compared with GridWay meta-scheduler – which does not support this feature. Finally, the Laxity is set between 0 and 10 minutes, being the average 5 minutes.

C. Performance evaluation

In this section a comparison between the SA-layer with both techniques of calculating the job execution time and the original GridWay meta-scheduler is outlined.

To evaluate the environment and the performance of both techniques of calculating the job execution time (using data logs or a linear function), several statistics can be used, described next. Scheduled job rate is the fraction of accepted jobs, i.e., those whose deadline can be met [8]. Loss rate is the inverse of scheduled job rate. QoS not fulfilled means the number of jobs rejected, plus the number of jobs that were initially accepted but their executions were eventually delayed. Thus, their QoS agreements were not fulfilled (the deadline was not met). These are measures of the QoS perceived by the user.

Furthermore, two more statistics are presented, namely overlap and waste. These ones are measures of system performance (meta-scheduler point of view), and they indicate how well the job completion time was estimated. Overlap records the number of minutes that a job execution is extended over the calculated estimation. Waste records the number of minutes that are not used to execute any job because the meta-scheduler thought that jobs would need more time to complete their executions. Both statistics are related to the accuracy of predictions, thus they do not make sense when there are not predictions – the original GridWay experiments.

The results of this evaluation are shown in Figure 5. The submission rate is from 1 to 4 jobs per minute, being the total number of jobs for each test 20, 40, 60 and 80, respectively.

Figure 5 (a) represents the number of jobs accepted because the meta-scheduler has enough free slots to allocate them meeting the requested QoS requirements. Using SA-layer, the more jobs there are in the system, the more lost jobs there are. On the other hand, using original GridWay, all jobs are always accepted because there is not a layer which decides whether or not is possible to execute the job meeting the deadline. For lower submission rates (20), the accepted job rate is the same for both ways of estimating the execution times. However, when the submission rate is higher (40, 60 and 80), using log data to estimate execution times yields better results, since it can accept more jobs because the estimation is better and different for each resource. This different estimation for each resource makes the prediction more accurate, thus the overlap and waste times are smaller than when the linear function is used (as can be seen in Figures 5 (c) and (d)). Furthermore, using log-based estimations allows more jobs to be accepted since fewer gaps need to be reserved.

Figure 5 (b) shows the number of jobs that were not executed with the QoS requested, and includes lost jobs and jobs executed out of the deadline. Again, the more jobs there are in the system, the more jobs not executed with the QoS requested there are. This is also true for original GridWay. Besides, GridWay performed worst than SA-layer. That is because in SA-layer some jobs were not accepted since the system estimated that it would not be possible to execute them with the QoS requested. On the other hand, as figure depicts, the log-based estimation exhibits better performance since the estimation is more accurate and adjusted for each resource.

Figures 5 (c) and (d) depict the mean overlap and waste times for the linear function and log-based job execution time estimations, respectively. There are not overlap and waste time for GridWay because it does not make any prediction about the job execution times. They depict a more accurate estimations when data log information is used since both overlap and waste are smaller. Having lower overlap time, there will be fewer jobs that do not meet the QoS requirements. This explains the results showed in Figure 5 (b). On the other hand, having lower waste time, more jobs can be accepted since each accepted job requires fewer reserved slots. This explains the results showed in Figure 5 (a).

Furthermore, if there is great heterogeneity among the power of resources, the linear function does not take into account such heterogeneity, thus the waste and/or overlap
times would be bigger. Finally, as can be concluded from Figure 5, it is a better option to use log data to estimate execution times, since this takes into account the heterogeneity of Grid resources. On the other hand, the results also highlight the importance of using scheduling in advance to get a higher QoS in a Grid environment.

VI. CONCLUSIONS AND FUTURE WORK

Several research works aim at providing QoS in Grids by means of advanced reservations. However, this scenario is not always feasible, since not all the LRMS permit reservations, or not all resources belong to the same administrative domain. Therefore, we proposed scheduling in advance (first step of advanced reservation) as a possible solution to provide QoS to Grid users.

In order to provide users with QoS, performing meta-scheduling in advance is very significant. With this type of scheduling, we can estimate whether a given application can be executed before the deadline specified by the user. But this requires to tackle many challenges, such as developing efficient scheduling algorithms that scale well. Another factor of great impact that needs to be studied is how to predict the execution time of jobs.

In this work, a comparison between using or not scheduling in advance is carried out. This comparison highlights the importance of using scheduling in advance to meet the QoS requested by users. Also, a comparison between a homogeneous (based on a linear function) and a heterogeneous (based on log data) way to estimate the job execution time is depicted. This comparison stresses the importance of taking into account the heterogeneity of Grid resources in
the estimation of job execution times. The log-based method calculates this time more accurately, thus executing more jobs meeting their deadlines.

Finally, recall that both the meta-scheduling in advance and advanced reservation in Grid environments are open fields that still need research since there are no definitive solutions (in terms of scalability and / or efficiency). Besides, many of the ideas developed to provide QoS in Grids have been evaluated in simulated environments, but our work is being carried out under a real environment.

As future work we want to differentiate the network transfer time from execution time of the jobs. Thus, the job execution time estimation is more accurate. In this way, it is possible to schedule more jobs by reserving only the slots needed for the execution time. Apart from this, job rescheduling must also be addressed, since it is needed whenever a resource leaves the Grid. In such case, the jobs scheduled to that resource have to be rescheduled to the available ones.

**ACKNOWLEDGMENT**

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grants “Consolider Ingenio-2010 CSD2006-00046”, “TIN2006-15516-C04-02” and “TIN2009-14475-C04-03”; jointly by JCCM and Fondo Social Europeo under grant “FSE 2007-15516-C04-02” and “TIN2009-14475-C04-03”; jointly by MEC and European Commission FEDER funds under grants available ones.

REFERENCES


