Using Network Information to Perform Meta-scheduling in Advance in Grids

Luis Tomás[1], Agustín Caminero[2], Blanca Caminero[1], and Carmen Carrión[1]

[1] Dept. of Computing Systems,
The University of Castilla-La Mancha, Spain
{luistb,blanca,carmen}@dsi.uclm.es
[2] Dept. of Communication and Control Systems,
The National University of Distance Education, Spain
accaminero@scc.uned.es

Abstract. In extremely heterogeneous and distributed systems, like Grid environments, it is quite difficult to provide quality of service (QoS). In addition, the dynamic behaviour of the resources makes the time needed to complete the execution of a job highly variable. So, fulfilling the user QoS requirements in a Grid is still an open issue. The main aim of this work is to provide QoS in Grid environments through network-aware job scheduling in advance. This paper presents a technique to manage idle/busy periods of resources using red-black trees which considers the network as a first level resource. Besides, no *a priori* knowledge on the duration of jobs is required, as opposed to other works. A performance evaluation using a real testbed is presented which illustrates the efficiency of this approach to meet the QoS requirements of users, and highlights the importance of taking the network into account when predicting the duration of jobs.

Keywords: Grid meta-scheduling, network, QoS, red-black trees.

1 Introduction

In a Grid environment the resources are in different domains under different access policies. This fact makes their search and use a hard task for users. Also, manually accomplishing this process is not feasible in a large-scale Grid environments with many potentially available resources. Hence, the Grid infrastructure must provide the needed services for automatic resource brokerage which take care of the resource selection and negotiation process [1]. This infrastructure is named "*meta-scheduler*" [2].

The user's experience of the Grid is determined by the functionality and performance of this meta-scheduler system. But the heterogeneous and distributed nature of the Grid along with the different characteristics of applications complicate the brokering problem. To further complicate matters, the broker typically lacks total control and even complete knowledge of the status of the resources [3].

One key idea to solve this problem is to ensure that a specific resource is available when the job requires it. So, it becomes necessary to reserve or schedule the use of resources in-advance [4]. *Reservation in advance* can be defined as a restrictive or limited delegation of a particular resource capacity for a defined time interval [5]. The objective

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 431–443, 2010.

of such reservation in advance is to provide quality of service (*QoS*) by ensuring that a certain job meets its QoS requirements.

The main challenge of reservations in advance is that it is difficult to decide whether a job can be executed fulfilling its QoS requirements without knowing the exact status of the resources in the future [6]. However, reservation in advance mechanisms enable QoS agreements with users and increase the predictability of a Grid system [7], at the expense of creating resource fragmentation [8].

As next sections depict, our work is different from the ones mentioned above because it is based on *meta-scheduling in advance* in Grids rather than reservations in advance, since reservation may not always be possible. This paper proposes a new *network-aware* algorithm to tackle the scheduling in advance problem. This algorithm is concerned with the dynamic behaviour of the Grid resources, their usage, and the characteristics of the jobs. This research focuses on low-cost computational heuristics that consider the network as a first level resource. This is needed because the network has a very important impact on the performance of jobs, as studied in [9,10,11,12], among others.

The usage of resources is managed by means of *red-black trees*. This idea has already been tried in [6,8], where authors assume that users have *a priori* knowledge on the job duration – which may not be true most times. So estimations on the completion time of jobs need to be calculated. To this end, it becomes necessary to predict the dynamic behaviour of the resources in the future. In the present work, estimations for job duration are calculated in two different ways: (1) using the Total Completion Time (`TCT`), and (2) Execution and Transfer Times Separately (`ETTS`), thus estimations on network transfer times have to be calculated. In (1), estimations on job durations use information of previous executions, and does not consider the network transfers (only completion times, that include transfer and execution times). In (2), estimations on the execution and transfer times of jobs are calculated independently. For execution time, an estimation is calculated similarly to the completion time in (1), and transfer times are calculated using bandwidth predictions through log data along with the number of bytes to transfer. Thus, both techniques pay attention to the heterogeneity of Grid resources and do not assume users have *a priori* knowledge on the duration of jobs, as assumed in [6,8]. These ways of estimating the job completion times are presented and evaluated in this paper.

The paper is organized as follows. Related work is presented in Section 2. In Section 3 a brief overview of the general network-aware meta-scheduling in advance framework is presented. Section 4 explains the extensions implemented to handle scheduling in advance. Section 5 presents the experiments carried out for evaluating the proposal. Finally, the conclusions obtained and the suggested guidelines for future work are outlined in Section 6.

2 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 [13]. 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 [14] and Legion [15].

Regarding the aforementioned advanced reservations of resources, Globus Architecture for Reservation and Allocation (GARA) [16] was introduced 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 for the manipulation of advanced reservation of different resources. Since then, advanced reservations have been studied in numerous contexts, such as clusters (*Maui Scheduler* [17]). 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. Another important system is *VIOLA* [18], which includes a meta-scheduling framework that provides co-allocation support for both computing and network resources. It allows the network to be treated as a resource within a meta-scheduling application.

Despite support for reservation in the underlying infrastructure is currently limited, a reservation in advance feature is required to meet QoS guarantees in Grid environments, as several contributions conclude [7,16]. Qu [19] describes a method to overcome this shortcoming by adding a Grid advanced reservation manager on top of the local scheduler(s). The performance penalty imposed by the usage of advanced reservations (typically decreased resource utilization) has been studied in [20]. 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 [21].

On the other hand, this work aims at performing scheduling in advance. This way of scheduling needs to perform predictions about the future network status and about job duration into resources. Currently, a broad range of estimation techniques are built around historical observations. In [22], it is shown that although load exhibits complex properties, it is still consistently predictable from past behaviour. In [23], an evaluation of various linear time series models for prediction of CPU loads in the future is presented.

3 Network-Aware 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 dynamicity 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.

Support for reservations in advance 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 [6].

A Grid reservation in advance process can be divided into two steps [5]:

1. **Meta-scheduling in advance:** Selection of the resources to execute the job, and the time period when the execution will be performed.
2. **Negotiation for resource reservation:** Consists on the physical reservation of the resources needed for the job, which may not always be possible.

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, 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 why our work is aimed at performing meta-scheduling in advance rather than reservations in advance in order to provide QoS in Grids. That is, the system keeps track of the meta-scheduling decisions already made in order to make future decisions. So, if only Grid load exist, this would be enough to provide QoS since the meta-scheduler would not overlap jobs on resources.

The algorithms for meta-scheduling in advance need to be efficient so they can adapt themselves to dynamic changes in resource availability and user demand without affecting system and user performance. Moreover, they must take into account resource heterogeneity since Grid environments are typically highly heterogeneous. For this reason, it could be useful to employ techniques from computational geometry to develop an efficient heterogeneity-aware scheduling algorithm [6].

In this work jobs do not have workflow dependencies and users provide both the input files and the application itself. Taking into account these assumptions an *scheduling in advance process* follows the next steps:

1. First, a "user request" is received. Every request must provide a tuple with information on the application and the input QoS parameters: (in_file, app, t_s, d). in_file stands for the input files required to execute the application, app. In this approach the input QoS parameters are just specified by the start time, t_s (earliest time jobs can start to be executed), and the deadline, d (time by which jobs must have been executed) [8].
2. The meta-scheduler executes a *gap search* algorithm. This algorithm obtains both the resource and the time interval to be assigned for the execution of the job.
3. If it is not possible to fulfill the user's QoS requirements using the resources of its own domain, communication with meta-schedulers from other domains starts.
4. If it is still not possible to fulfill the QoS requirements, a negotiation process with the user is started in order to define new QoS requirements.

In this process, the goodness of scheduling depends heavily on the quality of available information regarding the resources, but independence and autonomy of domains is another obstacle. This is because domains may not want to share information on the load of their resources. Moreover, in a Grid environment, resource contention causes host load and availability to vary over time, and makes the execution time predictions quite difficult [24].

The prediction information can be derived in two ways [24]: application oriented and resource oriented. For *application-oriented* approaches, the running time of Grid tasks is directly predicted by using information about the application, such as the running time of previous similar tasks. For *resource-oriented* approaches, the future performance of a resource such as the CPU load and availability is predicted by using the available information about the resource, and then such predictions are used to predict the running time of a task, given the information on the task's resource requirement.

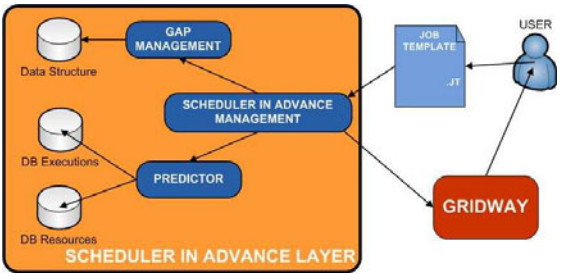


Fig. 1. The Scheduler in Advance Layer (SA-layer)

In our case we use a mixture between these two approaches. We use application-oriented approaches to sort out the execution time of the application and resource-oriented approaches to calculate the time needed to perform the network transfers.

4 A Framework for Network-Aware Meta-scheduling in Advance

In this section, the implementation carried out allowing network-aware meta-scheduling in advance is outlined. First, the structure of the framework is presented, followed by the policies for allocating jobs into gaps in resources. Next, the data structures used for managing this information are shown. Finally the prediction needs are discussed.

Our proposal is implemented as an extension to the GridWay meta-scheduler [2]. It is an intermediate layer, called *Scheduler in Advance Layer (SA-layer)*, between the users and the on-demand Grid meta-scheduler, as Figure 1 depicts. The *SA-layer* is a modular component that uses functions provided by GridWay in terms of resource discovery and monitoring, job submission and execution monitoring, etc., and allows GridWay to perform network-aware meta-scheduling in advance. The SA-layer stores information concerning previous application executions (called *DB Executions*), and the status of resources and network over time (called *DB Resources*). Moreover, a new parameter has been added to GridWay, named `JOB_INFORMATION`. In this new parameter the user may indicate some information about the job. First, if the user knows the input and output size, he sets this information. After that, the user may set other characteristics related to the jobs, such as job arguments, which enable a more accurate estimation for the job execution time. On that purpose, the execution time of jobs in a given resource is estimated by using prediction. This prediction takes into account the characteristics of the jobs, the power of the CPU of the resources and the network future status. By processing these information about applications and resources, a more accurate estimation of the job completion time in the different computational resources can be performed. Besides, the memory overhead is negligible (about several Mbits).

4.1 Gap Management

In this implementation, resource usage is divided into time slots. Then, we have to schedule the future usage of resources by allocating the jobs into the resources at one

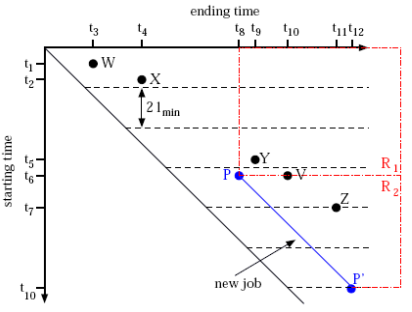


Fig. 2. Idle periods regions [6,8]

specific time (taking one or more time slots). For this reason, allocation policies (carried out by *Gap Management* module in Figure 1) to find the best slots for each job, data structures (represented by *Data Structure* in Figure 1) to keep a trace of slots usage and algorithms for estimations on job durations (implemented by *Predictor* in Figure 1) are needed, which are explained the next.

The job allocation influences how many jobs can be scheduled due to the 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 consecutive allocations.

In our first approach, a *First Fit* policy has been considered. This technique selects the first free gap found that fits the new job. It can create 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 leaves less free time slots after allocating the job. The created fragments are smaller, but it is harder to use those free slots to allocate new jobs.

4.2 Data Structure

The data structure used to keep track of the free time slots is a key aspect. A suitable data structure yields better execution times and reduces the complexity of algorithms. Furthermore, the data structure will also influence on the scalability of the algorithm.

That is the reason why the structure used in this work is *red black trees* [6,8]. The objective of using these type of trees is to develop techniques to efficiently identify feasible idle periods for each arriving job request, without having to examine all idle periods.

This data structure is managed by the *Gap Management* module (see Figure 1). This module represent the information of tree data structure in a geometrical way. So, each job is represented by a single point in the plane as Figure 2 [6,8] depicts. The job coordinates are *starting time* and *ending time*. Labeled points represent the idle periods (gaps) with its start and finish time. P represents the earliest start and end times, whilst P' represent the latests, for the current job. Thus, the line between P and P' represents the periods when this new job can be scheduled. All the points above and to the right of this line represent possible gaps to allocate the job.

As Castillo explains in [8], the trees can be divided into two regions, named R1 and R2, as Figure 2 [6,8] 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.

A job scheduled in an idle period will create at most two new idle periods: one between the begining 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. So, this region is searched first. Work on studying other ways of searching the regions is among the future work.

4.3 Predictor

Predictions of job execution time are quite difficult to obtain since there are performance differences between Grid resources and their performance characteristics may vary for different applications. Techniques for such predictions include applying statistical models to previous executions [22] and heuristics based on job and resource characteristics [11,23]. Based on this, the algorithm proposed by Castillo [6,8] is extended to take into account the heterogeneity of Grid resources.

In the present work, estimations for the duration of jobs are calculated based on (1) using the Total Completion Time of jobs (`TCT`), and (2) Execution and Transfer Times Separately (`ETTS`). In (1), the mean completion time of previous executions of similar applications on the selected host is used to manage idle/busy periods on the resource. This may lead to poor resource usage since a resource may be considered as busy when in fact the job is being transferred – the job execution has not started yet. Besides, in (2), separate execution and transfer times are used, which improves the resource usage. For the execution time, an estimation is calculated similarly to the completion time in (1) – this includes execution and queueing times. Regarding transfer times, the mean bandwidth of the day before for the time interval in which the job will be allocated is calculated. Using this information, along with the total number of bytes to transfer, the time needed to complete the transfers is estimated.

Estimating execution and transfer times separately yields more accurate predictions, which in turn lead to better utilization of resources and better QoS delivered to users. This is because the meta-scheduler knows for each time-slot if a job is actually being executed at a resource or being transferred to it, which allows the meta-scheduler to manage idle/busy periods of computing resources more efficiently.

Thus, both techniques pay attention to the heterogeneity of Grid resources and do not assume that users have *a priori* knowledge on the duration of jobs, as assumed in [6,8]. These two ways of estimating the completion time of jobs are presented and evaluated in this paper. Also, in both cases, 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. Please note that two applications are considered to belong to the same application type when they have the same input and output parameters – in terms of number, type and size.

5 Experiments and Results

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

5.1 Testbed Description

The evaluation of the scheduling in advance implementation has been carried out in a local real Grid environment (depicted in Figure 3). The testbed is made of resources located in two different buildings belonging to The University of Castilla La-Mancha (UCLM). In one building there are, on the one hand, one machine which carries out the scheduler tasks, and on the other hand, several computational resources. In the second building, there is a cluster machine with 88 cores. 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.

5.2 Workload Used

One of the GRASP [25] 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. The output file is sent to the *"result node"*. 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 considered 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; *"T_Exec$_i$"* is the time needed to execute 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

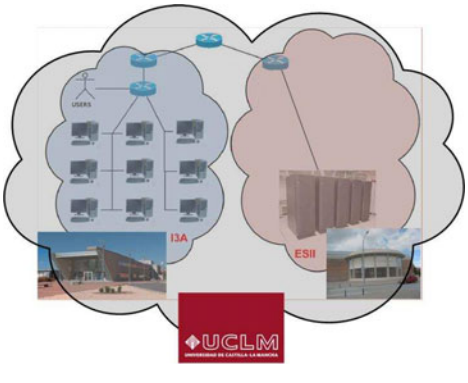


Fig. 3. Grid testbed topology

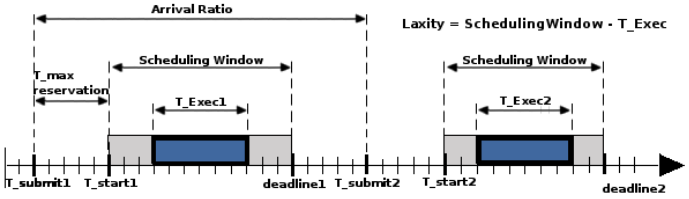


Fig. 4. Workload characteristic

sent; and *"Laxity"* represents how strict the user is when scheduling a job, which is the difference between the *"Scheduling Window"* and the *"T_Exec"* 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 up to 1 hour, with an average of 30 minutes. The *Laxity* is set between 0 and 10 minutes, being the average 5 minutes. The submission rate is from 1 to 4 jobs per minute. Finally, time slots last 1 minute. The results shown are the average of 5 executions for each case.

5.3 Performance Evaluation

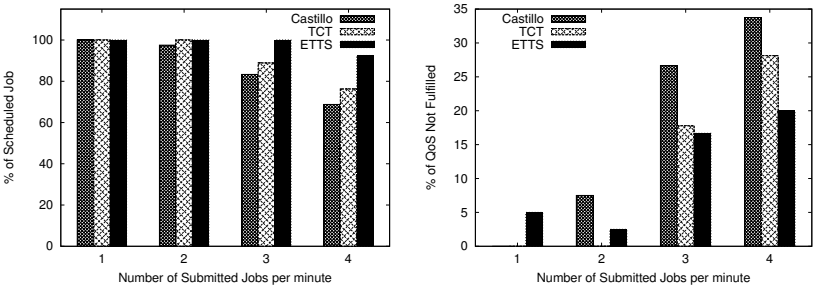
In this section, a comparison between the SA-layer with both techniques of calculating the job execution time and a straight-forward implementation of the algorithm proposed by Castillo et al. [6,8] is outlined. Our proposal has already been compared with the GridWay original framework (without support for scheduling in advance), resulting in a performance gain [26]. Now, this paper compares our framework with the original framework developed by Castillo et al. [6,8], and shows (1) there is no need to have an a priori knowledge on the execution time of jobs – as long as predictions can be made; and (2) the importance of performing estimations on transfer and execution times separately, rather than taking both parameters altogether.

To evaluate the importance of using network information in the meta-scheduling process, several statistics can be used. *% of Scheduled job rate* is the fraction of accepted jobs, i.e., those whose deadline can be met [8]. *% of 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.

From the point of view of the meta-scheduling system, there is another statistic, namely *waste*. It records the number of minutes inside reservations (not physical reservation) that were not used since the meta-scheduler thought that jobs would need more time to complete their executions. This statistic is related to the accuracy of predictions.

Results from the user and system point of view are depicted in Figures 5 and 6, respectively. In these plots, estimations on the Total Completion Time are labelled as `TCT`, while estimations with Executions and Transfer Times Separately are labelled as `ETTS`. The results obtained when using the algorithm proposed by Castillo et al. [6,8] are labelled as `Castillo`.

First, Figure 5 (a) represents the percentage of scheduled jobs (the meta-scheduler has enough free slots to allocate them, meeting the QoS requirements). The more jobs



(a) Jobs Scheduled (b) QoS not Fulfilled

Fig. 5. Estimating total completion time or separate estimations for execution and transfer times

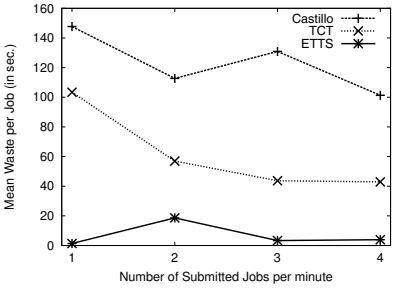


Fig. 6. Estimations waste time

there are in the system, the more lost jobs there are. All the algorithms have a similar behaviour at low loads. The differences appear when the system load becomes close to saturation (3 jobs/min.). Notice that when using ETTS, lost jobs appear only when the load goes over 3 jobs/min. Moreover, the loss rate in this case is very small compared with the other two techniques. At this load, Castillo lost around 31 % of the scheduled jobs but ETTS only around 7.5 %. So, using ETTS to estimate execution times yields better results, since it can accept more jobs because the estimation is more accurate.

Figure 5 (b) shows the percentage of jobs that were not executed with the QoS requested, and includes lost jobs and jobs completed beyond the deadline. Again, the more jobs there are in the system, the more jobs not executed with the requested QoS there are. For lower submission rates (1 and 2 jobs/min.), it is not essential to make separate estimations for executions and transfer times since there are enough free slots. Thus, reserving the number of slots in such a tight way does not show any enhancement. However, for higher submission rates (3 and 4 jobs/min.) it becomes very important to make this difference (ETTS line) since a noticeable reduction in the number of lost jobs is achieved.

Figure 6 depicts the mean waste times in calculating the job completion time estimations. This graphic highlights that even having a greater number of running jobs, the waste is lowest when using ETTS than when using TCT, that is, without estimating

execution and network times separately in the gap reservation. With 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). Also, resource utilization is better since there is less wasted time in between executions of jobs. So, the resource will be idle for less time.

6 Conclusions and Future Work

Several research works aim at providing QoS in Grids by means of *advanced reservations*. However, making reservations of resources is not always possible. Sometimes not all the LRMS permit reservations, while in other cases not all the resources belong to the same administrative domain. There are even other types of resources which may belong to several domains at the same time, such as network bandwidth. So, we proposed scheduling in advance (first step of advanced reservation) as a possible solution to provide QoS to Grid users.

This type of scheduling allows to 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 or how to predict the jobs execution time. For this reason, the prediction of the Grid resources status is essential.

In this work, a comparison between using estimations on the Total Completion Time (`TCT`) and Execution and Transfer Times Separately (`ETTS`) is presented. Also, both techniques are compared with an implementation of the scheduling in advance algorithm proposed by Castillo et al. [6,8]. This comparison highlights the importance of calculating network estimations independently because it improves the resource usage, thus allowing more jobs to be scheduled. 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.

The development and implementation of new efficient and scalable algorithms is one of the challenges of this research. So, among the future work we are planning to include new parameters such as *trust of resources*. This parameter could be measured as the historical percentage of jobs assigned to a computational resource that do not get their QoS requirements. Another challenge is work on *jobs rescheduling* to provide the specified QoS in Grid. When the scheduler fails to allocate a job, it is possible to allocate new incoming jobs by rescheduling existing already scheduled jobs whenever possible without affecting their QoS (*Replanning Capacity* [6]).

Acknowledgments

This work was supported by the Spanish MEC and MICINN, as well as European Commission FEDER funds, under Grants “CSD2006-00046” and “TIN2009-14475-C04”. It was also partly supported by JCCM under Grants “PBI08-0055-2800” and “PII1C09-0101-9476”.

References

1. Yahyapour, R.: Considerations for resource brokerage and scheduling in Grids. In: Proc. of Parallel Computing: Software Technology, Algorithms, Architectures and Applications (PARCO), Dresden, Germany (2003)
2. Huedo, E., Montero, R.S., Llorente, I.M.: A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. Future Generation Computing Systems 23(2), 252–261 (2007)
3. Elmroth, E., Tordsson, J.: An interoperable, standards-based grid resource broker and job submission service. In: Proc. of the 1st Intl. Conference on e-Science and Grid Computing (e-Science), Washington, DC, USA (2005)
4. Sulistio, A.: Advance Reservation and Revenue-based Resource Management for Grid Systems. PhD thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia (2008)
5. GWD-I, Global Grid Forum (GGF): Advance reservations: State of the art. J. MacLaren (2003), `http://www.ggf.org`
6. Castillo, C., Rouskas, G.N., Harfoush, K.: Efficient resource management using advance reservations for heterogeneous grids. In: Proc. of the Intl. Parallel and Distributed Processing Symposium (IPDPS), Miami, USA (2008)
7. Siddiqui, M., Villazón, A., Fahringer, T.: Grid capacity planning with negotiation-based advance reservation for optimized QoS. In: Proc. of the 2006 Conference on Supercomputing (SC 2006), Tampa, USA (2006)
8. Castillo, C., Rouskas, G.N., Harfoush, K.: On the design of online scheduling algorithms for advance reservations and QoS in grids. In: Proc. of the Intl. Parallel and Distributed Processing Symposium (IPDPS), Los Alamitos, USA (2007)
9. Tomás, L., Caminero, A., Caminero, B., Carrión, C.: Studying the Influence of Network-Aware Grid Scheduling on the Performance Received by Users. In: Proc. of the Grid computing, high-PerformAnce and Distributed Applications (GADA), Monterrey, Mexico (2008)
10. Tomás, L., Caminero, A., Caminero, B., Carrión, C.: Improving GridWay with Network Information: Tuning the Monitoring Tool. In: Proc. of the High Performance Grid Computing Workshop (HPGC), Hold Jointly with the Intl. Parallel & Distributed Processing Symposium (IPDPS), Roma, Italy (2009)
11. Caminero, A., Rana, O., Caminero, B., Carrión, C.: Performance evaluation of an autonomic network-aware metascheduler for Grids. Concurrency and Computation: Practice and Experience 21(13), 1692–1708 (2009)
12. Tanwir, S., Battestilli, L., Perros, H.G., Karmous-Edwards, G.: Dynamic scheduling of network resources with advance reservations in optical grids. Int. Journal of Network Management 18(2), 79–105 (2008)
13. Farooq, U., Majumdar, S., Parsons, E.W.: Efficiently scheduling advance reservations in grids. Technical report, Carleton University, Department of Systems and Computer Engineering (2005)
14. The Globus Alliance (2009), `http://www.globus.org`
15. Legion Project (2009), `http://legion.virginia.edu/`
16. Roy, A., Sander, V.: GARA: A Uniform Quality of Service Architecture. In: Grid Resource Management, pp. 377–394. Kluwer Academic Publishers, Dordrecht (2003)
17. Maui Cluster Scheduler (2009), `http://www.clusterresources.com/products/maui/`
18. Waldrich, O., Wieder, P., Ziegler, W.: A meta-scheduling service for co-allocating arbitrary types of resources. In: Proc. of the 6th Intl. Conference on Parallel Processing and Applied Mathematics (PPAM), Poznan, Poland (2005)

19. Qu, C.: A grid advance reservation framework for co-allocation and co-reservation across heterogeneous local resource management systems. In: Proc. of 7th Intl. Conference on Parallel Processing and Applied Mathematics (PPAM), Gdansk, Poland (2007)
20. Smith, W., Foster, I., Taylor, V.: Scheduling with advanced reservations. In: Proc. of the 14th Intl. Parallel and Distributed Processing Symposium (IPDPS), Washington, DC, USA (2000)
21. Wieczorek, M., Siddiqui, M., Villazon, A., Prodan, R., Fahringer, T.: Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid. In: Proc. of the 2nd Intl. Conference on e-Science and Grid Computing (e-Science), Washington, DC, USA (2006)
22. Dinda, P.A.: The statistical properties of host load. Scientific Programming 7(3-4), 211–229 (1999)
23. Jin, H., Shi, X., Qiang, W., Zou, D.: An adaptive meta-scheduler for data-intensive applications. Intl. Journal of Grid and Utility Computing 1(1), 32–37 (2005)
24. Zhang, Y., Sun, W., Inoguchi, Y.: Predict task running time in grid environments based on CPU load predictions. Future Generation Computing Systems 24(6), 489–497 (2008)
25. Chun, G., Dail, H., Casanova, H., Snavely, A.: Benchmark probes for grid assessment. In: Proc. of 18th Intl. Parallel and Distributed Processing Symposium (IPDPS), Santa Fe, New Mexico (2004)
26. Tomás, L., Caminero, A., Carrión, C., Caminero, B.: Meta-Scheduling in Advance using Red-Black Trees in Heterogeneous Grids. In: Proc. of the High Performance Grid Computing Workshop (HPGC), Hold Jointly with the Intl. Parallel & Distributed Processing Symposium (IPDPS), Atlanta, USA (2010)