<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper published in *Future generations computer systems*. This paper has been peer-reviewed but does not include the final publisher proof-corrections or journal pagination.

Citation for the original published paper (version of record):

Conejero, J., Caminero, B., Carrion, C., Tomas, L. (2014)

From volunteer to trustable computing: Providing QoS-aware scheduling mechanisms for multi-grid computing environments.

Future generations computer systems, 34: 76-93

<http://dx.doi.org/10.1016/j.future.2013.12.005>

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-88266>

From Volunteer To Trustable Computing: Providing QoS-Aware Scheduling Mechanisms for Multi-Grid Computing Environments

Javier Conejero^a, Blanca Caminero^a, Carmen Carrión^a, Luis Tomás^b

^a*Computing Systems Department
University of Castilla-La Mancha
Campus Universitario s/n
02071 Albacete, Spain*

^b*Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden*

Abstract

The exploitation of service oriented technologies, such as Grid computing, is being boosted by the current service oriented economy trend, leading to a growing need of Quality of Service (QoS) mechanisms. However, Grid computing was created to provide vast amounts of computational power but in a best effort way. Providing QoS guarantees is therefore a very difficult and complex task due to the distributed and heterogeneous nature of their resources, specially the volunteer computing resources (e.g., desktop resources).

The scope of this paper is to empower an integrated multi QoS support suitable for Grid Computing environments made of either dedicated and volunteer resources, even taking advantage of that fact. The QoS is provided through SLAs by exploiting different available scheduling mechanisms in a coordinated way, and applying appropriate resource usage optimization techniques. It is based on the differentiated use of reservations and scheduling in advance techniques, enhanced with the integration of rescheduling techniques that improve the allocation decisions already made, achieving a higher re-

Email addresses: `Francisco.Conejero@uclm.es` (Javier Conejero),
`mariablanca.caminero@uclm.es` (Blanca Caminero), `Carmen.Carrion@uclm.es`
(Carmen Carrión), `luis@cs.umu.se` (Luis Tomás)

source utilization and still ensuring the agreed QoS. As a result, our proposal enhances best-effort Grid environments by providing QoS aware scheduling capabilities.

This proposal has been validated by means of a set of experiments performed in a real Grid testbed. Results show how the proposed framework effectively harnesses the specific capabilities of the underlying resources to provide every user with the desired QoS level, while, at the same time, optimizing the resources' usage.

Keywords:

Service Level Agreement, Grid, QoS, Scheduling in Advance, Reservation, Rescheduling

2010 MSC: 68-02

2010 MSC: 68-M14

2010 MSC: 68-M15

2010 MSC: 68-M20

1. Introduction

The exploitation of service oriented technologies, such as Grid and high performance computing environments, is being boosted by the current service oriented economy trend [1], leading to a growing interest of Quality of Service (QoS) mechanisms [2]. However, Grid computing was conceived to provide vast amounts of computational power, of a wide variety and geographically dispersed resources, but in a best effort way. Therefore, providing QoS guarantees is a very difficult and complex task due to the distributed and heterogeneous nature of their resources, specially the volunteer computing resources that can be available (e.g., desktop resources). Moreover, this fact becomes even more complicated when considering efficient resource usage and user prioritization.

Service Level Agreements (SLAs) have become the key element to support and empower QoS in these environments, but they have also become necessary in order to enforce service utilization policies and usage commitments. SLAs can be defined as service agreements between users and service providers, in which all expectations and obligations of a service are explicitly defined [3]. Therefore, they are used as formal method between the users and the infrastructure to define the required QoS. Moreover, SLAs let the system know what performance the user expect for their jobs, which can also be used to help the system to perform smarter scheduling decisions.

In this work, SLAs with different service levels are designed in order to allow the users to define their requirements in a better way. Users define their QoS requirements in high-level SLA metrics that are mapped to the low-level resource metrics used by the proposed framework within Grid environments.

Unfortunately, SLAs do not provide autonomous QoS handling mechanisms. This task lays on the infrastructure manager, which needs to map the requirements agreed on the SLAs with its infrastructure services and delegate them to an automatic resource brokerage, most known as *scheduling systems* [4]. Generally, Grid resources are owned by different entities (e.g. users, Universities, etc.) and in some cases, these resources are only available when they are not being used by their owners. They are also known as *volunteer resources* and sometimes belong to the Grid and other times not. On the contrary, there may be other resources, known as *dedicated resources* which, despite being shared amongst different users, present a more reliable performance since their usage may usually be reserved and usually remain as part of the Grid. This dynamic behavior in the resource availability makes

difficult to develop an efficient scheduling algorithm to avoid SLAs violations. Consequently, there is a need of a SLA management framework able to establish usage commitments with the users and handle the resources through the available scheduling mechanisms in order to maximize the resources usage and fulfill users expectations.

The proposal presented in this work tackles this complex issue by making use of different scheduling in advance algorithms in a combined way within the same framework depending on the commitments established on the SLAs. More precisely, Reservation in Advance (RA) and Meta-Scheduling in Advance (MSA) approaches have been used as the basis of the proposal.

RA can be defined as a limited delegation of a particular resource capability over a defined time interval [5]; and MSA can be defined as the first step of the RA process, that is, the resource selection for the defined time interval, but without performing any physical reservation of its usage. These two approaches face the scheduling problem from different points of view. RA tries to ensure specific resources on a determined time window, while MSA focuses on keeping track of the scheduled jobs and resource usage in order to take future scheduling decisions. These two mechanisms can be complemented with prediction algorithms (that can be used to forecast the resources performance and act consequently) and rescheduling algorithms (which try to reorganize the schedule at run time).

The main purpose of this paper is to empower an integrated multi QoS support within Grid computing environments where different types of resources (volunteer and dedicated ones) presenting different reliability and confidence belong to the system. This is provided through SLAs, exploiting the aforementioned different QoS mechanisms in a coordinated way, in order to guarantee the agreed QoS in terms of timing and reliability. Subsequently, resource usage is optimized by applying the most appropriate technique to support any desired QoS level of performance.

In order to build a trustable computing environment over an infrastructure that it is not (i.e., it is based on volunteer resources), the framework proposed in [6, 7] for multiple QoS levels and SLA support within Grid environments has been enhanced. The above described QoS mechanisms are mapped to each QoS level, in order to exploit them in an organized and coordinated way. The most trustable resources (dedicated ones), with the stronger technique (RA), are used when the QoS required is higher and conversely when lower. Besides, the framework has been extended with the implementation of rescheduling techniques, to further increase resource usage

and facilitating the provision of the different levels of QoS offered.

To summarize, the main contributions of this paper are: a) the orchestration of different scheduling and reservation in advance algorithms together with predictive techniques in order to match the user requirements; b) the use of rescheduling techniques to reduce the fragmentation of the system, maximizing the amount of SLAs processed, and minimizing the number of SLA violations, whilst at the same time improving the ratio of jobs that can be processed with higher QoS requirements; and finally c) a comprehensive evaluation of the framework in a real Grid testbed.

The structure of this paper is as follows. In Section 2, the sample scenario (with the problem of QoS integration) to be faced is presented whilst some background about the QoS mechanisms this work is focused on are outlined in Section 3. Then, Section 4 defines the proposal of this work, namely, how to integrate and manage multiple QoS mechanisms in a coordinated way. Section 5 describes the reference architecture used in order to make a proof-of-concept implementation of the proposal. Subsequently, performance evaluation results are presented in Section 6, and related work is presented in Section 7. Finally, conclusions and suggested guidelines for future work are outlined in Section 8.

2. Sample Scenario

Providing QoS guarantees to users is a very difficult and complex task, specially within Grid environments that includes volunteer resources. Its distributed and heterogeneous nature, together with the variety of resources that can be available (and that may leave the Grid the next moment), the user expectations and the lack of total control over the resources, difficult the optimal exploitation of all resources. In order to tackle this issue there is a wide variety of QoS mechanisms (such as: meta-schedulers, negotiation, reservations, security, etc.) that have been deeply researched. But, each one has been designed to act at a different level and with different purposes.

Figure 1 presents a generic Grid environment for a better understanding of their architecture and usage workflow. Grid environments are composed of a collection of computational resources and a pool of users who wish to run their applications or jobs on the Grid. Among those resources, some of them are dedicated (more reliable) while others are shared in a volunteer way, belonging to the Grid at one moment and being able to leave it at any time. The first challenge is to establish an agreement about the service that

Figure 1: Sample Scenario.

the user wants and the one that the system is able to provide. Then, the next challenge is how to manage the jobs into the Grid to ensure their performance and avoid any kind of agreement violations. Furthermore, achieving high utilization by better using the resources to make possible the acceptance of more requests is also a challenging task.

This is therefore a problematic scenario for enhanced QoS provisioning, specially when dealing with volunteer resources within the Grid infrastructure. Moreover, users do not have a uniform behavior. The amount of requests arriving to the Grid can be very varied, in terms of frequency and requirements. They must be attended and processed, offering the best QoS possible. However, different requests may need to be differentiated and their peculiarities determined in order to know which mechanism can empower the QoS that every particular request needs.

To address these challenges, we present a novel differentiated QoS aware framework considering multiple SLAs for service usage enforcement in Grid systems. SLAs represent the first QoS mechanism that any request can find in a Grid environment. They are used to specify the user expectations in

terms of suitable metrics to evaluate their fulfillment, and establish a formal contract which both (user and service provider) agree. But Grid environments shared nature, that depends on multiple and independent resource owners with unpredictable behavior, difficulties their management. Thus, SLAs are mainly used to enforce the agreed utilization terms although there are a number of issues that still have to be addressed, such as penalties, billing, provider trusting, third party monitoring services, incomplete or unclear service specifications, etc. Then, thanks to the information provided by the users on those SLAs, better QoS can be provided. By using the information regarding how strong the QoS requirements are, the scheduling action can be carried out in a smarter way, providing more QoS assurance when needed and using other less reliable resources when not. That is, SLAs are used as a hint on the performance expected by users.

Right after SLA management in the stack, schedulers become the next QoS mechanism (which is monitoring dependent). Schedulers are designed to handle user's requests. They implement scheduling algorithms in order to exploit resources in an optimized way. However, not all the schedulers are prepared to provide support for all terms defined on an SLA (as they can be very varied) and a translation from SLA terms into computer understandable metrics needs to be addressed. As a consequence, smarter schedulers or another kind of mechanisms are needed in order to provide support for the SLA requirements. Furthermore, the monitorization also needs to be improved in order to provide support for new metric definition.

One of the most important terms that usually appear on an SLA are the time constraints (very common within application workflows). There are two main scheduling mechanisms to support time constraints: Meta-Scheduling in Advance (MSA) and Reservation in Advance (RA).

Both of them depend on the environment, since Grid middlewares do not implement these mechanisms by default (they can be found as external modules) [8]. Consequently, they have to be deployed in order to exploit them. MSA and RA tackle the same problem, which is to plan and control future executions, but from a different point of view and by means of implementing different algorithms. They can be found on different levels within Grid environments, as MSA is usually located over the middleware and RA within each Local Resource Management System (LRMS).

Prediction and rescheduling appear as other mechanisms that can complement RA and MSA. Rescheduling focuses on resource scheduling reorganization in order to improve the performance of the whole system. And

prediction pushes for future knowledge of the resources in order to evaluate in advance their availability, performance, etc. There is a wide variety of algorithms for both of them [9, 10, 11]. In addition, QoS mechanisms like policy, security, redundancy and fault-tolerance, can also be found very important since they are designed to increase the guarantees on each job execution.

To summarize, there are several mechanisms at our disposal in order to improve and ensure the QoS. But there is a lack of interaction among them by definition because they focus on different points of view to tackle QoS provision. The objective of this work is to exploit the available scheduling mechanisms in a combined and coordinated way.

3. Background

This work focuses on: Service Level Agreements (SLAs), Reservation in Advance (RA), Meta-Scheduling in Advance (MSA) and Rescheduling; and how to exploit them at the same time in a coordinated way. Hence, this Section presents a brief survey of these mechanisms individually, emphasizing the advantages and disadvantages for each one.

3.1. Service Level Agreements

One of the most QoS-enabling mechanism nowadays is SLAs. Its main objective is to provide the QoS negotiation feature, between users and service providers, in a standardized and formal way.

SLAs can be defined as contracts between users and service providers, in which all expectations and obligations are explicitly defined [3]. They represent a legal statement of the QoS expectations established on each SLA that both participants of the negotiation expect to achieve and the responsibilities and obligations that both participants assume.

There have been many efforts within the SLA topic. Among them, it must be highlighted the Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) of the Open Grid Forum (OGF) effort on SLAs standardization, which resulted on the WS-Agreement specification [12, 13], which can nowadays be considered the most important. Therefore, this work is focused on the WS-Agreement specification.

However, the WS-Agreement specification only defines the structure and negotiation protocol. As a consequence, SLAs do not provide autonomous QoS handling mechanisms, nor detailed information about the terms that may appear on an SLA.

WS-Agreement considers a simple negotiation protocol (Figure 2a) on its first specification, which starts with a query from the user to the service provider and it responds with an SLA *template/s* (plain SLA structure without term values). On receiving the templates, the user fills the terms with the requested needs (formalizing the SLA) and sends it to the service provider. Subsequently, the service provider checks if it is properly formed and takes the decision on acceptance or rejection of the SLA. For both decisions, the user receives a message with the result of the decision: a rejection message if rejected or an acknowledgement message (ACK) with the corresponding endpoint reference (EPR) if accepted.

Figure 2: WS-Agreement negotiation protocol.

The WS-Agreement extension introduces the renegotiation process into the negotiation protocol (Figure 2b). The main difference is that this protocol enables a loop between user and service provider on which the SLA can be polished with an offer-counteroffer model. Consequently, it adds more flexibility into the negotiation process.

WS-Agreement considers the *templates* as open contracts, where the service provider specifies the service terms that are offered. However, some restrictions must be respected, such as the structure. The structure defined by WS-Agreement for *templates* and SLAs consists of four blocks:

1. *Name*: Where the name of the agreement is specified (for human identification).
2. *Context*: All agreement details are specified (e.g., client and provider ids).
3. *Service Terms*: It contains all terms related to assurances and required commitments.
4. *Guarantee Terms*: With the QoS values expected.

In addition, another block appears in *templates*, named *Constraints*, which is used by the service provider to specify all the creation constraints.

In our work, some templates for specifying the user QoS have been defined and a negotiation protocol according to WS-Agreement is used.

3.2. Reservations in Advance

The aim of Reservations in Advance (RA) is to provide QoS by ensuring that resources are available when jobs need them (for defined time periods) [5].

The reservation process can be divided into two main steps:

1. *Selection* of the resources where the job will be executed. This stage requires to control the future schedule of each resource.
2. *Reservation*, i.e., to perform the physical reservation of the resources needed for the job.

This is a mechanism implemented within LRMS and widely used within cluster environments, where it is particularly useful (due to the local management of these resources) for applications with huge resource requirements and that need a concurrent guaranteed allocation of resources. It excels as a resource assurance feature for specific time windows and even for internal resources characteristics, e.g., amount of RAM. PBS [14], Torque [15] and Sun Grid Engine (SGE) [16] are good examples of cluster schedulers that provide support for reservations.

However, reservation cost is higher than traditional mechanisms for batch queuing jobs [17]. Moreover, when extrapolated to Grids, reservations are not always feasible as this feature is not available in all resources associated to the Grid. Thus, their usage (if available) may be limited by the resource owner, as it is necessary to have specific permissions to use this feature. And even with permission, it can be policy dependent; this means that the priority is subject to the resource owner policy. Other disadvantage inherent

to the usage of reservations is fragmentation. This happens in non adjacent requests scenarios.

Our work is not focused on a specific scheduler, as we provide a customizable handler that can perform reservations in any LRMS.

3.3. Meta-Scheduling in Advance

Meta-Scheduling in Advance (MSA) focuses on the first step of the RA process, where resources are selected and their future schedule is controlled. But it does not perform any physical reservation of any resource, resulting in a non invasive mechanism.

As a consequence, MSA has high adaptability (it can work with any kind of resource associated to the Grid) within a Virtual Organization (VO).

With MSA, resources are not ensured, and the MSA system has more flexibility and is able to adapt to any dynamic change of the Grid environment. Profiling, reschedule and forecast mechanisms therefore become really important in order to improve the whole system performance.

MSA can be seen as a light version of RA due to the fact that this mechanism allows to perform modifications on the scheduling on-the-fly. Depending on the margins between the user requirements and the real execution time (profiled or forecasted), the schedule can be optimized in such a way that resource usage keeps as high as possible, as reservations are not performed.

As a drawback, MSA cannot ensure resource availability (as they are conditioned by the resources owner usage, such as: heavy processes load). Thus, the QoS cannot be ensured with the same level of guarantee as with RA.

SA-Layer [18] enacts a pure implementation of this QoS mechanism, so we focus on it.

3.4. Rescheduling

Rescheduling is a feature that can be available within scheduling systems. The objective of this feature is to reorganize the schedule by performing multiple (re)scheduling processes in order to optimize its performance. In order to do that, this feature considers the migration of jobs between resources and time. It can work easily within MSA, as well as with RA, provided that job QoS requirements can be fulfilled with another resource that is not explicitly specified.

On the contrary, the rescheduling process includes a computational overhead, that increases as the environment and the scheduling needs grow. Hence, rescheduling needs to be carefully applied.

The rescheduling algorithm can be triggered by:

- *Periodical events*: The rescheduling process can be performed periodically in order to keep an optimized schedule.
- *On demand events*: Rescheduling may be necessary when the fragmentation is over a defined threshold, when resource usage is unbalanced, or triggered by any other circumstance, such as a job rejection.
- *Unexpected events*: Undesirable events can force the need of a rescheduling process (e.g., resource disconnection, resource performance drop, network failure, etc.).

Rescheduling can be enabled in the latests versions of the SA-Layer [18].

3.5. Other QoS mechanisms

There are other QoS mechanisms that must be highlighted, such as forecasting, profiling and policy definition. These mechanisms can be used within MSA, RA and Rescheduling in order to improve or condition their performance.

Policies help service providers to prioritize jobs and/or resources (such as fair-share usage [19]). Forecasting and profiling can be used in order to collect more information about the jobs, their executions, their behavior within resources or users requests. Therefore, this information can be useful to optimize the scheduling and even to improve the policies. This mechanisms are even more interesting within self-tuning systems.

Furthermore, there are other QoS mechanisms not related to scheduling which are also important, such as security, redundancy and fault-tolerance. These mechanisms empower the reliability of the QoS, but they are out of the scope of this work.

4. Proposal

The scope of this paper is to empower an integrated QoS support within an infrastructure that was build to provide computing power in a best-effort way, as it includes volunteer resources amongst others. This is addressed

through SLAs by exploiting different available scheduling mechanisms, with the objective of improving and/or ensuring the agreed QoS. In order to achieve this, how the different QoS mechanisms are used and interact are defined.

As reviewed in Section 2, the problems tackled in this article are very complex and need to be addressed carefully. Not all mechanisms can be used, other cannot be used together, and what is more important, it is necessary to ensure that the QoS mechanisms do not negatively interfere among them.

The proposal presented in this work focuses on SLAs, Reservation in Advance (RA), Meta-Scheduling in Advance (MSA) and Rescheduling (and how to exploit them at the same time) by extending the framework proposed in [6]. In addition, the mechanisms inherently provided by the middleware are used, such as: resource discovery, scheduling and security.

SLAs are a static mechanism to interact with the users and translate their expectations in terms of QoS into low level requirements with the main objective of service terms enforcement, whilst RA and MSA are mechanisms that provide support for the jobs associated to each SLA in scheduling terms. RA and MSA differ on how to handle these jobs looking for the best QoS as possible. Therefore, we propose to make them work together in a coordinated way, as they are not originally designed to work together and have different performance, behavior and QoS guarantees.

To achieve this goal, a multi-level QoS schema based on time requirements is used, whilst SLAs are used to specify the user requirements and classify them among these levels of QoS. Moreover, these requirements are mandatory in order to use RA and MSA.

The terms that can be specified are: start-time (t_S) and deadline (t_D). The t_S and t_D terms are used to specify the time window for executing the job associated to each SLA, although the job will likely need less time than specified by this time window. If the execution does not respect this parameters, the SLA is automatically regarded as not fulfilled or violated.

As a multi-level QoS schema, another parameter to define the confidence on its execution (regarding the importance on the job that the user expects) is needed, as t_S and t_D terms only specify when the job has to be processed.

Our proposal uses the three level QoS classification based on the confidence that the user expects for the execution of the job associated to each SLA proposed in [6] independently of the amount of users and jobs at the same time.

The three level QoS confidence are defined with the *Bronze*, *Silver* or *Gold*

labels. The values of the time parameters for the three level QoS proposal are:

- *Weak* $\rightarrow$ $t_S = 0$ $t_D = \infty$ $QoS = Bronze$
- *Moderate* $\rightarrow$ $t_S = t_{1Start}$ $t_D = t_{1Deadline}$ $QoS = Silver$
- *Strong* $\rightarrow$ $t_S = t_{2Start}$ $t_D = t_{2Deadline}$ $QoS = Gold$

The lowest QoS service (*Weak*) corresponds to a best-effort service, without temporal restrictions. For the jobs associated to this QoS level, the start-time can be omitted or set to zero (as it will be executed as soon as possible) and the user does not care about the deadline. So, the deadline in this case is infinity and it can also be omitted. This does not mean that they will last forever, the sense of deadline is that the user is not concerned about when the job finishes. So, the jobs associated to this QoS level can be scheduled on low confidence Grid resources, such as the ones whose usage can not be physically reserved (e.g. desktops resources). Anyway, a really loose deadlines (i.e., a day) are internally defined for those jobs to avoid some starvation.

For the *Moderate* and *Strong* QoS levels, the start-time and deadline are set by the user to their needs, but the difference between them is the expected confidence on its execution. It can be *Moderate* or *Strong*, depending on the user needs for the reliability expected from the service for each job: the *Moderate* level takes care of the service but does not totally ensure the availability of the resource where the execution is scheduled; on the contrary, the *Strong* level ensures the resource availability by only using high confidence resources associated to the Grid, i.e., those that support RA mechanisms.

Moreover, as presented in [6] and assuming that no resource is switched off, this QoS differentiation with RA mechanisms ensure that Gold accepted requests will be executed meeting the time constraints, but at expenses of fewer jobs been accepted. To try to solve this problem, in this work we propose the inclusion of rescheduling techniques that allow the system to accept a greater amount of jobs, whilst at the same time providing (at least) the same confidence.

The inclusion of these rescheduling mechanisms has an impact in all QoS levels, since it focuses on the reorganization of the jobs associated to each SLA. But for our proposal we only consider rescheduling for *Weak* and *Moderate* QoS levels (namely, Bronze and Silver SLAs correspondingly), since one of the objectives is to ensure the achievement of *Strong* QoS level SLAs

(namely, Gold SLAs). Consequently, with the usage of this mechanism we expect to improve resource usage, as well as to increase the acceptance rate prioritized by QoS level. But due to the fact that our proposal distinguishes three kind of QoS levels (consequently, three kind of SLAs) and two kinds of resources (high confidence and low confidence resources), the rescheduling needs to be tuned accordingly (Figure 3).

The amount of levels is not increased with the inclusion of rescheduling techniques, since it performs internal optimizations that are transparent to the user, with the objective of providing a better QoS and resource utilization. Furthermore, this fact helps on the evaluation of the performance between proposals, and allows to analyze the impact of the new mechanism in conjunction with MSA and RA.

Figure 3: Modifications

Gold SLAs are not considered in the rescheduling process (unlike Silver and Bronze SLAs) due to the fact that the system performs physical reservations in advance for this kind of SLAs, so Gold SLAs can not be disturbed by other kind of SLAs. Each Gold SLA accepted is mapped to a resource with RA capability (1), reserved for the time requirements established, on which the associated job execution is processed. Therefore, the RA system guarantees the availability of the resource for the accepted Gold SLAs. Moreover, since Gold SLAs can only be mapped to resources with RA capability, the amount of this kind of resources limits the amount of Gold SLAs that

can be accepted and processed with the same time requirements, avoiding a Gold SLAs flooding by rejecting them instead of accepting and violating their agreed terms. Consequently, rescheduling is focused on Silver and Bronze SLAs. In addition, if too many users are requesting Gold QoS, but not enough resources are available, besides rejecting the requests, the system may offer the users to schedule the jobs with a less strong QoS level, such as silver SLA. But in order to exploit this mechanism renegotiation becomes necessary, which is considered for future work.

The jobs associated to Silver SLAs can be processed in any resource associated to the Grid. But the SA-Layer has been modified to be able to check high confidence resources first (2), and in case of an impossibility of allocating the job on this kind of resources, check low confidence resources (3), where, according to the prediction techniques, is feasible to execute it fulfilling their time constraints. Accepted Silver SLAs can be rescheduled, but in order to keep the priorities defined for the three QoS levels, the rescheduling process is conditioned. When the job associated to a Silver SLA has been successfully allocated on a high confidence resource, this may be affected by two factors: first, the *Gold SLA acceptance*, and second, the *availability of high confidence resources*. In the first case, the job associated to a Silver SLA is rescheduled if it has been allocated on a resource with RA capability and an incoming Gold SLA (higher priority) that needs the same resource for the same time requirements collide. So, the rescheduling mechanism tries to move the job associated to the Silver SLA to another high confidence resource (4a) in order to allow the system to allocate the new job associated to the Gold SLA. In the second case, if there are no available resources on the high confidence group, the jobs associated to Silver SLAs can be scheduled into low confidence resources (4b). Every job associated to Silver SLAs that has been scheduled into low confidence resources can be rescheduled in resources from the same group (4c). They are not rescheduled within high confidence resources, since these resources have been previously checked. It must be noted that to be able to move a Silver SLA associated job, the QoS agreed must be kept. This means that a Silver SLA associated job is moved if (and only if) it can be reallocated in another resource/time still fulfilling the QoS requirements agreed. This limitation on the resources where Silver SLA associated jobs should be rescheduled improves the performance of the rescheduling process since the amount of resources to check is delimited.

Finally, the jobs associated to Bronze SLAs can only be processed in low confidence resources (low confidence, e.g., volunteer resources) (5). And the

- WS-Agreement compliant SLA support for Grids.
- Flexibility for new terms and policy definitions.
- Meta-scheduling in Advance feature (SA-Layer) [18].
- Based on Globus Toolkit 4 [8] and GridWay 5.6 [20].

The top layer, named SLA-Manager, has the responsibility of interacting with users and transfer their queries to the SLA-Backend layer. This interaction is performed by a WS-Agreement compliant SLA negotiation process. As a consequence, the interoperability and compatibility between users and service providers is assured.

The SLA-Backend layer's main objective is to take the acceptance or rejection decisions on the requests performed by the users through the SLA-Manager. It is also responsible of managing and monitoring the jobs and system in order to fulfill the accepted SLAs. So it can be considered as a high level scheduler that is able to control the layers below it.

The support of MSA is addressed by the SA-Layer [18], which provides support for scheduled jobs in that way by using the GridWay metascheduler.

All the stack leans on Globus Toolkit 4 (GT4) middleware, which is considered the '*de facto*' standard middleware in Grid computing. Furthermore, the SLA-Manager and SLA-Backend are built as GT4 web services, so they can exploit all the advantages provided by GT4, such as: standards, interfaces, security (GSI), etc.

Finally, the LRMS, which can be found within each Grid computing node, manages the execution locally. There are many different LRMS, so it is possible to find different capabilities across the Grid (e.g., Torque [21] or PBS [14], which can provide RA capability).

As integration of multiple QoS mechanisms regards, the two top layers take the decisions, control the job executions and users interaction. On the other hand, the lower layers are in charge of how to schedule the jobs into the resources and the execution itself. They are described in the next subsections with greater detail.

5.1. SLA-Manager

The top layer of the framework is the SLA-Manager, whose objective is to provide mechanisms for SLA negotiation between users and service providers following the WS-Agreement 1.0 specification [12].

As outlined in Subsection 3.1, WS-Agreement considers the templates as open agreements, which have to be customized by the service providers. Templates are designed by the service provider, and the terms and limitations of an agreement are explicitly defined on them. When they are filled by the user with the desired terms, and the negotiation protocol is started, they are considered as SLAs.

For our purposes, these templates require explicitly the parameters described in Section 4 in order to characterize the three QoS levels previously defined on each template (Figure 5). The user must fill these terms with the desired *start-time*, *duration* and the *QoS level* expected. These parameters are considered mandatory and they have to appear in Gold and Silver SLAs, while *start-time* and *duration* can be omitted for Bronze QoS level since they have a best-effort treatment.

Internally, the SLA-Manager focuses on offering a WS-Agreement compliant interface and delegates the decision of acceptance or rejection to the next layer, namely SLA-Backend (which performs the scheduling for each SLA job for the given time requirements). Consequently, in order to keep track of the SLAs status, the SLA-Manager monitors the status of each job associated to each SLA in terms of time requirement fulfillment. As a result, the SLA-Manager keeps track of the status of all SLAs for the agreed terms (i.e., amount of accepted, rejected and violated SLAs, performance ratio, etc.).

Furthermore, in order to ease the interaction between users and the SLA-Manager, there are two interfaces: CLI and Web Interface. The CLI allows users to interact with the SLA-Manager from a command prompt, allowing them to design and develop automated submission scripts. On the other hand, the Web interface offers a graphic interactive mechanism (Figure 6) that allows users to submit their jobs in a simpler way (hiding the complexity of the templates).

5.2. SLA-Backend

The SLA-Backend is responsible for the acceptance/rejection decision and meta-scheduling of the incoming jobs, taking into consideration the terms specified on each SLA (e.g., OS, CPU architecture, RAM, cost, etc.).

The SLA-Backend layer receives all the requests that the SLA-Manager receives. To take the acceptance or rejection decisions, policies need to be implemented in this layer, which allows an easy policy definition, modification or update. Thus, the SLA-Backend also supports independent term policies.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsag:Template xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  wsag:TemplateId="BasicTemplate">

  <wsag:Name>Basic Template</wsag:Name>

  <wsag:Context>
    <wsag:AgreementInitiator xsi:type="xsd:string">User</wsag:AgreementInitiator>
    <wsag:AgreementResponder xsi:type="xsd:string">GridI3A</wsag:AgreementResponder>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <wsag:ExpirationTime>2012-12-01T14:00:00</wsag:ExpirationTime>
    <wsag:TemplateId>1</wsag:TemplateId>
    <wsag:TemplateName>Basic Template</wsag:TemplateName>
  </wsag:Context>

  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm wsag:Name="Job" wsag:ServiceName="Job@I3A">
        <jsdl:JobDefinition>
          <JobDescription>
            <jsdl:ApplicationName>Job</jsdl:ApplicationName>
          </JobDescription>
        </jsdl:JobDefinition>
        <wsag:JobTimeDescription>
          <JobStartTime>2012-12-01 10:00:00</JobStartTime>
          <JobDuration>240</JobDuration>
        </wsag:JobTimeDescription>
      </wsag:ServiceDescriptionTerm>

      <wsag:GuaranteeTerm wsag:Name="ServiceQuality" wsag:Obligated="ServiceProvider">
        <wsag:ServiceScope wsag:ServiceName="Job@I3A"/>
        <wsag:ServiceLevelObjective>
          <wsag:KPITarget>
            <wsag:KPIName>ServiceQuality</wsag:KPIName>
            <wsag:CustomServiceLevel xsi:type="xsd:string">
              GOLD <!-- SILVER - BRONZE -->
            </wsag:CustomServiceLevel>
          </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
      </wsag:GuaranteeTerm>
    </wsag:All>
  </wsag:Terms>

  <wsag:CreationConstraints>
    <!-- Nothing yet -->
  </wsag:CreationConstraints>

</wsag:Template>

```

Figure 5: Template example

The SLA-Backend layer stands over the SA-Layer since all jobs are managed through it. Consequently, it keeps track of the jobs thanks to the SA-Layer and shares this information with the SLA-Manager. The SLA-Backend takes high level decisions whilst the SA-Layer handles the jobs in terms of time scheduling. When a job can be scheduled (it is notified to the SLA-Manager and the SLA is accepted), the SLA-Backend submits the associated job to the SA-Layer which handles the job execution in terms of start-time and deadline. Moreover, the SLA-Backend informs about the job status when the SLA-Manager requires it in order to keep track of the SLAs status. Alternatively, when the system can not allocate a job (even after the rescheduling process) the SLA-Backend notifies it to the SLA-Manager and

The image shows a web interface for SLA Job Submission. At the top, there is a red header bar with the UCLM logo on the left, the title "SLA Job Submission" in the center, and the A2CA logo on the right. Below the header, the text "Fill the form:" is displayed. The form contains several input fields: "Job:" with a dropdown menu, "Start Date:" with a date input field (27-02-2013) and a format hint (dd-mm-yyyy), "Start Time:" with a time input field (12:23:00) and a format hint (hh:mm:ss), "Duration:" with a number input field (120) and a unit hint (Seconds), and "Quality of Service:" with a dropdown menu showing options: Gold, Gold, Silver, and Bronze. A "Submit" button is located to the right of the Quality of Service dropdown. At the bottom of the form, there are logos for I3A (Instituto de Investigación en Informática de Albacete) and the Universidad de Castilla-La Mancha. The footer of the page is a red bar with the text "Universidad de Castilla-La Mancha".

Figure 6: SLA-Manager Web Interface

the SLA is rejected.

More in detail, the SLA-Backend internal structure is modular. It is composed of three main blocks, two databases and three scripts (Figure 7). The *Request Manager* module is the interface between the SLA-Manager and the other two modules: *Brain* and *Job Submission*. Decision policies are implemented in the *Brain* module. Consequently, the *Backend* checks the *Brain* module and evaluates the SLA requirements with the system status in order to take decisions. The *Job Submission* module is in charge of handling the jobs once they are accepted by using the *scripts* (*Submit*, *Cancel* and *Monitor*), which are in contact with the SA-Layer.

Ultimately, the *Jobs* and *Status* databases are used to store the information of the jobs that have been accepted, the system status for the terms that the SLA-Backend has correspondingly meta-scheduled.

The SLA-Backend is able to perform physical reservations on the LRMS that provide RA capability. In this case, the SLA-Backend submits the job to the SA-Layer denoting where it is going to be executed after performing the reservation (done through the *Reservation Handler* module). If the reservation can not be performed or the job can not be scheduled, the SLA-Backend

Figure 7: SLA-Backend internal structure

notifies the SLA-Manager this circumstance in order to reject the SLA.

Originally, rescheduling can not be handled as the reorganization performed by this technique affect on the job traceability, and the SLA-Backend needs this information in order to keep the consistency for a proper job management. As a consequence, the SLA-Backend has been tuned in order to keep track of where each job is being processed, by improving the *monitoring* module to check the movements that the SA-Layer performs on the schedule. But not all jobs can be rescheduled (e.g., Gold SLAs associated jobs), so the *Reservation Handler* has been improved to notify the SA-Layer their non reschedulable condition.

These modifications allow the system to reschedule the jobs between resources, respecting the three QoS levels established in the proposal, in order to optimize the Grid exploitation and keep the guarantee agreed on each SLA.

5.3. SA-Layer

The meta-scheduling in advance support is implemented in the SA-Layer [18] as a layer over the GridWay meta-scheduler [20]. It has been designed as a GridWay extension that provides scheduling in advance autonomously, at-

tending to the SLA-Backend requests, and using the resource discovery, job delegation and monitorization (for resources and jobs) capabilities provided by Gridway. It also implements forecasting algorithms, rescheduling mechanisms and resources trust.

Internally, the SA-Layer divides the resources time in intervals, called *slots*. The system therefore schedules the future resource usage by mapping the incoming requests with them for the given time requirements (using one or more adjacent slots). This scheduling process requires an efficient data structure. To this end, red-black trees [22] are used due to their efficiency on locating adequate slots and, as a consequence, providing efficient insert and delete operations. Furthermore, the SA-Layer stores information related to the behavior of previous jobs and resources, which is used for forecasting purposes. These predictions are used to try to prevent unexpected situations such as not enough available capacity at the chosen resource, which could lead to not meeting the requested deadlines.

The SA-Layer (which implements the functionality described in Subsection 3.3) also includes different rescheduling techniques, as presented in [23]. They are in charge, for instance, of reallocating the jobs of a resource that has failed or left the Grid into the other available resources.

In this work we use and adapt the on demand rescheduling functionality provided by SA-Layer, which is triggered when the MSA algorithm is not able to map the requests of a job. This means that the SA-Layer checks its schedule in order to determine if there is any free resource for every request during a the given time window, and if this is not possible, the rescheduling process is performed in order to identify a suitable resource for the request. Therefore this mechanism can be categorized into the *reactive techniques* as it is only applied when a job allocation fails.

This rescheduling process is detailed in Algorithm 1. When a job request is rejected, the rescheduling technique is started (line 8). The first step then is to select the suitable candidate jobs (already scheduled ones) for being reallocated leaving their previously reserved slots to the new incoming job (line 9). Then, the system filters out the jobs whose reserved slots are not into the target time interval of the new incoming job. The remaining jobs are sorted by taking into account their probability of being rescheduled (still fulfilling their QoS agreements), from more to less probable. This is measured as their associated laxity, obtained through Equation 1.

Algorithm 1 Rescheduling Algorithm

```
1: Let  $job$  = the new incoming job
2: Let  $R$  = set of resources known to GridWay  $\{R_1, R_2, \dots, R_n\}$ 
3: Let  $laxity\_threshold$  = the threshold used to filter resources
4: Let  $SelectJ$  = set of already scheduled jobs  $\{J_1, J_2, \dots, J_n\}$  which have
   reserved slots in between the start time and the deadline of the  $job$ , and
   whose laxity is above  $threshold$ 
5: Let  $SortJobsByLaxity(J)$  the function which sorts the jobs of the list
    $J$  by its probability of being reallocated, from bigger to smaller laxity
   values
6: Let  $Gap^R$  = set of available gaps in  $R$ 
7: Let  $TS^{J_i, R_i}$  = time slots reserved for job  $J_i$  in resource  $R_i$ 
8: if  $job$  allocation fails then
9:    $SelectJ$  = filterJobs( $laxity\_threshold$ )
10:   $SelectJ$  =  $SortJobsByLaxity>SelectJ$ )
11:  for each  $SelectJ_i \in SelectJ$  do
12:    if  $TS^{SelectJ_i, R_i}$  is feasible for  $job$  then
13:      for each  $Gap_k^R \in Gap^R$  do
14:        if  $Gap_k^R$  is feasible for  $SelectJ_i$  then
15:          Allocate  $SelectJ_i$  at  $Gap_k^R$ 
16:          Allocate  $job$  at  $TS^{SelectJ_i, R_i}$ 
17:          Exit
18:        end if
19:      end for
20:    end if
21:  end for
22: end if
```

$$Laxity = \frac{SchedulingWindow - ExecutionTime}{ExecutionTime} \quad (1)$$

where $SchedulingWindow$ is the time interval in which the job has to be executed ($deadline - startTime$), and $ExecutionTime$ is the time the system estimated that the job would need to complete its execution for the resource and the time interval where it was allocated. Once we have estimated their laxities, we first check the jobs whose associated laxity is greater since they have more chances of being rescheduled.

Once the list of target jobs is obtained ($SelectJ$ in Algorithm 1), the system checks if the number of slots of $SelectJ_i$ ($TS^{SelectJ_i, R_i}$) is enough for allocating the incoming job. In that case, the system tries to allocate the already scheduled job ($SelectJ_i$) in other resource and/or time interval (line 14), and if possible, its previous reserved slots are used for allocating the new incoming job (line 16). This way both jobs are allocated achieving a better resource utilization.

However, this rescheduling algorithm is unaware of the QoS agreements made at the SLA-Manager level. Consequently, the SA-Layer has been improved in order to take into account which jobs can be moved and among which kind of resources.

With this aim, the SA-Layer splits the available resources by taking into account if they can be physically reserved or not (higher or lower confidence). It then makes the scheduling allocation process able to allocate the jobs in one or another group. Therefore, the SLA-Backend indicates the QoS requirements by specifying which group or resources to search. Then, the SA-Layer only searches the resources providing physical reservation for the Gold SLA jobs, whilst for the Bronze SLA jobs only the resources that do not provide it. For Silver SLA ones, it tries to find a resource that provides physical reservations, and if not possible, searches then the rest of resources.

On the other hand, as rescheduling actions concern, the SLA-Backend also indicates which jobs cannot be moved (the Gold SLA jobs). Moreover, the SA-Layer avoids to move jobs from non-confident resources (the one that cannot be physically reserved) to confident ones (the one that provide the RA capability) in order to follow the schema proposed by the SLA framework explained in Section 4.

6. Evaluation

In this section, an evaluation of the proposal described in Section 4 over the architecture described in Section 5 is shown.

The Grid environment and the description of the experiments is detailed in the next subsection. Then, the workload is described and right after, the experimental results are analysed.

6.1. Grid environment

The proposal described in the present work is evaluated in the RAAP (UCLM High Performance Networks and Architectures Research Group)

Machine	CPU	Freq.	Main Mem.	HDD	Nodes	LRMS
Desktop 1	Amd Opteron 244	1.8 Ghz	1 Gb	140 Gb	1	Fork
Desktop 2	Amd Opteron 244	1.8 Ghz	1 Gb	80 Gb	1	Fork
Desktop 3	Amd Opteron 244	1.8 Ghz	1 Gb	80 Gb	1	Fork
Desktop 4	Amd Opteron 244	1.8 Ghz	1 Gb	80 Gb	1	Fork
Desktop 5	Amd Opteron 244	1.8 Ghz	1 Gb	80 Gb	1	Fork
Desktop 6	Amd Opteron 244	1.8 Ghz	1 Gb	80 Gb	1	Fork
Desktop 7	Intel Pentium 4	3.2 Ghz	3 Gb	200 Gb	1	Fork
Desktop 8	Intel Pentium 4	3.0 Ghz	2 Gb	260 Gb	1	Fork
Desktop 9	Intel Pentium D	3.4 Ghz	1 Gb	250 Gb	1	Fork
Cluster 1	Dual Amd Opteron 280	3.0 Ghz	8 Gb/node	500 Gb	3	Fork + Torque

Table 1: Grid nodes details.

Grid environment, which is composed of nine desktop nodes and a cluster. Details of this environment are shown in Table 1.

All of the Grid nodes belonging to the testbed are geographically spread, within different networks and being used frequently by their owners (user shared resources). The cluster associated to the RAAP Grid has 20 computing nodes, but 17 of them are dedicated to other purposes.

Moreover, all Grid nodes have GT4 Fork LRMS, which provides basic functionality. The cluster also implements Torque. Thus, it provides the RA capability.

6.2. Workload

In order to perform the evaluation, several users submit a set of SLAs (over a period of time) to the framework. As each SLA is used to determine the job, time restrictions and confidence that the user is expecting to be fulfilled, the job needs to be defined. The job associated to all SLAs for the evaluation is the *3node* benchmark [24]. This benchmark consists of sending a file from a *source node* to a *computation node*, which performs a search

Distribution	Gold	Silver	Bronze
A	$\sim 20\%$	$\sim 40\%$	$\sim 40\%$
B	$\sim 20\%$	$\sim 50\%$	$\sim 30\%$
C	$\sim 20\%$	$\sim 60\%$	$\sim 20\%$
D	$\sim 20\%$	$\sim 70\%$	$\sim 10\%$

Table 2: Workload distributions.

pattern, generating an output file with the number of successes.

The rationale for choosing that benchmark is due to the fact that it has parameterizable options to make it more computing intensive, and/or network demanding; allowing the definition of very different kind of jobs (from very light jobs up to heavy ones in terms of network and/or CPU usage).

The *3node* benchmark job parameters are set to 10 Mb input files for network transmission and random processing scale, with random size output files which can reach between one and ten times the size of the input file.

The SLA submission rate is random with a limit of 6 SLAs every 120 seconds (barely less than the maximum rate of productivity measured for the used Grid tested) during (at least) 2 hours of experiment. As resources are shared with users, experiments are repeated 5 times (at different times of the day across multiple days) in order to achieve representative results. And the time restrictions are also set randomly between 0 and 120 minutes (from SLA submission) for the *start-time* and between 10 and 20 minutes for the *deadline*. So, each experiment duration takes more than 4 hours.

The job parameters and submission rate represents the clients usage model of the system, which for the current Grid infrastructure is a high load stress test.

The environment is tested with four different workload distributions, which are determined by the amount of SLAs of each confidence type (set randomly by the user) (Table 2).

We keep the amount of Gold QoS level jobs between scenarios, as it represents an amount of SLAs that must be fulfilled independently of what happens with other QoS levels, letting enough temporal periods to be exploited by Silver and Bronze QoS level jobs. The rationale to set the amount of Gold SLAs to $\sim 20\%$ and keep it is due to the fact that there is only one resource where they can be processed within the evaluation environment.

Furthermore, in order to perform a deeper study of the environment and evaluate the behavior of the QoS mechanisms integration within the pro-

positional, all the experiments are also done with three different QoS policies or scenarios:

No-QoS) Without QoS levels, nor rescheduling (only applying MSA).

QoS) With QoS levels but without rescheduling (MSA + RA).

QoS+ReS) With QoS levels and rescheduling (MSA + RA + Rescheduling).

The first scenario (No-QoS) is used to evaluate the behavior of the Grid environment just applying the MSA mechanism. In the second scenario (QoS), the classification in levels and the RA mechanism is enabled (as proposed in [6]). Finally, within the QoS+ReS scenario, the entire proposal presented in this work is applied and evaluated.

6.3. Results

The results obtained are shown in this subsection, focusing on the behavior of the whole Grid system in terms of SLA rejection and violation rates. Besides that, the behavior of the reservation and scheduling in advance mechanisms are also analyzed. It must be noted that there are no Bronze SLA violations, nor rejection, due to their lack of time restrictions. So the behavior of the system for Bronze SLAs is analyzed in terms of waiting times.

The first result that must be highlighted is the validation of the proposal over the Grid. All mechanisms are deployed and their integrated use comes possible with this proposal. Moreover, an improvement of the performance and behavior is obtained.

The system has shown that it is able to reject any client submission if the expected QoS requirements can not be guaranteed in advance. So, when the system receives a higher amount of SLAs than it can fulfill, it starts rejecting SLAs.

First of all, we analyze the behavior of the system for the No-QoS policy, that is, without QoS levels nor RA nor rescheduling (just MSA), in terms of rejection and violation rates from the total SLAs submitted (Figure 8). Figure 8a shows the SLA rejection rate, that is, the amount of SLAs that were not accepted by the system (not establishing a contract between user and service provider), while Figure 8b depicts the SLA violation rate, which shows the amount of SLAs whose associated job could not be fulfilled with the conditions established on each SLA. The rejection rate is lower than 2%

Figure 8: No-QoS system behavior.

in all cases, for the given conditions. This means that the system accepts almost all of the received SLAs. On the contrary, the violation rate increases as the amount of Silver SLAs grows, reaching more than 4% of the accepted SLAs when the amount of Silver SLAs represents 70% of the total SLAs submitted (distribution D). Furthermore, Gold SLAs violations are produced in a representative way compared with the violations on Silver SLAs (e.g., receiving 20% Gold and 50%-60% Silver SLAs (D and C), the amount of Gold SLA violations is similar to the amount of Silver SLAs violations, but it is higher when the system receives 20% Gold and 40% Silver SLAs (A)).

By introducing the classification into QoS levels (scenario labeled as QoS), and focusing on the rejection rate (Figure 9a), it can be seen that there is a rejection increase compared with the No-QoS scenario (Figure 8a). This behavior is due to the fact that the classification in levels establishes limits on the resources where each SLA can be processed. Consequently, this limits impose a reduction on the number of nodes that can be used for a given QoS SLA level (e.g., Gold SLAs can only be mapped to resources with RA, while Bronze SLAs can be mapped to the rest and Silver SLAs can be mapped to any resource). Due to the fact that we have only one resource with RA at our disposal, when no more reservations can be performed on that resource, the Gold SLAs can not be guaranteed, hence no more Gold SLAs are accepted.

As a consequence, the rejection rate of Gold SLAs is higher than in No-QoS scenario. Furthermore, it increases as the amount of Silver SLAs grows, although the amount of Gold SLAs keeps constant. This happens because of the design of the classification in levels, as it is designed to map Silver SLAs to high confidence resources first, and due to the fact that the amount

Figure 9: QoS system behavior.

of Silver SLAs increases, the system allocates these resources, limiting the allocation of Gold SLAs.

Another curiosity occurs between distributions A and B within QoS scenario (Figure 9a), where both have a similar rejection rate, but with a different proportion of Silver and Gold SLAs. In distribution A, there is a 20% Gold and 40% Silver workload distribution, while in B there is a 20%-50% respectively. It could be expected a higher Silver and not lower Gold rejection rate in B, but the system would not be able to ensure the Gold successful processing if the amount of Silver SLAs is incremented (since they share the cluster). Consequently, there is a higher Gold rejection rate and this allows more Silver SLAs to be processed in the cluster (reducing the Silver rejection rate). Furthermore, the Silver rejection rate is also reduced because there are less Bronze SLAs without time requirements (desktops are more available to process Silver SLAs).

Regarding the violation rate (Figure 9b), the system is able to process all Gold SLAs without a violation. On the contrary, Silver SLAs suffer more violations since a higher amount of them are processed within desktop nodes (less reliable).

As a result of integrating QoS classification into levels and RA, two problems are identified: first, the priority of resource mapping for Silver SLAs has a harmful impact on the Gold SLAs rejection rate; second, although the system is able to ensure all Gold SLAs, the overall violation and rejection rates obtained are increased for the Silver SLAs. Reallocating the Silver jobs to other resources by making use of the rescheduling mechanism described in Section 3.4 will avoid these situations.

Figure 10: QoS+ReS system behavior.

The integration of the rescheduling mechanism (QoS+ReS) avoids a big amount of SLA violations (Figure 10b) (improving the whole system behavior in all cases), minimizing the number of violations to the lowest rate achieved in previous scenarios (No-QoS and QoS). Furthermore, it also helps to reduce the rejection rate (Figure 10a).

In the QoS+ReS scenario (Figure 10), there is an acceptance rate between 96% and 98% of all received SLAs (with the given conditions), and the maximum violation rate reached is 1.5%.

It is very important to observe that rescheduling does not affect to the Gold SLA violation rate, which is kept at 0% (Figure 10b). This is one of the objectives pursued, and ensures that Gold SLAs are completely fulfilled.

Furthermore, the integration of rescheduling, respecting the QoS classification into levels with MSA and RA, does not compromise the accepted Gold SLAs integrity (with the policy described in Section 4). Hence, this mechanism does not interfere on ensuring the Gold SLAs fulfillment.

Focusing in detail on the SLAs rejection behavior (Figure 11a), the introduction of the rescheduling mechanism (QoS+ReS line) reduces the rejection compared with not using it (QoS line). On the other hand, focusing more on detail in the SLAs violation behavior (Figure 11b), it is observed that without any kind of control (No-QoS line), the system executes more jobs meeting their time constraints (but without QoS differentiation among them) than when limiting the resources on where the jobs associated to the SLAs can be processed (due to the QoS classification (QoS line)). Moreover, enabling the use of rescheduling techniques within the QoS classification (QoS+ReS line) results in a dramatically improvement of the system behavior.

Figure 11: Detailed SLAs behavior.

The results obtained are quite impressive: a violation rate lower than 2% for the highest submission rate of SLAs with temporal restrictions (D = 20% Gold+70% Silver), while keeping the Gold SLAs at 0% violation rate and a rejection rate under 4.1%.

Looking deeper on the insights of the rescheduling mechanism for the performed tests (Figure 12a), the maximum rescheduling rate has been achieved for C and D distributions with 31% of the accepted jobs being rescheduled. This means that the system has no more margin to optimize the job schedule for the given tests.

In distribution A it can be observed that the Grid can successfully process almost all jobs without moving too many jobs (with very low rejection rate and very low violation rate). On the other hand, as the amount of Silver SLAs is increased, keeping constant the Gold SLA rate, the system is pushed with a higher stress level. As a consequence, the amount of rescheduled jobs is increased.

The rescheduling process can be very heavy and some jobs can suffer a high number of replanifications delaying their execution indefinitely. This is the case of Bronze SLA associated jobs, which do not have time restrictions, so they may suffer waiting delays. In order to determine if this is happening in any of the previously defined distributions, the number of rescheduling actions performed per job (assuming that 100% is the total number of rescheduling actions done) is analyzed (Figure 12b).

In distribution A, 80% of the jobs that were replanned did not suffer a second rescheduling action, 14% of the rescheduled jobs suffered 2 rescheduling actions, 5% suffered up to 3 rescheduling actions and the rest more.

Figure 12: Rescheduling behavior.

As the amount of Silver SLAs is increased and consequently, the Bronze SLAs reduced, the amount of more-than-once-scheduled jobs reduces. This is due to the time windows imposed by Silver SLAs and the fact that there are less Bronze jobs, thus reducing the chances of finding a suitable job candidate to be rescheduled. The percentage of multiple rescheduled jobs is less than 1%, therefore long delays over Bronze SLAs are not being experienced, even in the worst case for Bronze SLAs (competing against a high number of more prioritized SLAs).

Once we have verified that the Silver and Gold jobs are processed within the requirements established on their SLA, it is time to analyze the performance obtained by less prioritized jobs (associated to Bronze SLAs). In order to determine the behavior of the system with this kind of jobs, we have measured the waiting time to be processed (Figure 13). This behavior shows that Bronze jobs do not wait indefinitely. More in detail, in Figure 13, vertical lines determine the maximum and minimum waiting times observed from all processed Bronze SLA jobs, and the boxes determine the 50% medium observed. This means that 50% of the jobs had to wait within that range of time, while 25% of the jobs had to wait less and the 25% remaining, more time to be executed.

The observed waiting times are very close to each other in terms of top and bottom extremes. But there is a big reduction on the medium waiting time experienced by enabling QoS classification in levels (Figure 13(b)) and with rescheduling (Figure 13(c)), as the amount of Bronze SLAs decreases. Waiting time is reduced in general, although there are some jobs that have to wait longer, but no more than when not applying any QoS mechanism.

(Figure 13(a)).

When allowing the QoS classification in levels and applying rescheduling, the waiting time for Bronze SLAs associated jobs is considerably reduced (Figure 13(b & c)). When not using any technique (Figure 13(a)), 75% of the jobs have to wait more than 40 minutes to be processed. But with the QoS mechanism (and also with rescheduling) in place, 75% of them have to wait less than 30 minutes.

Figure 13: Bronze SLAs waiting time per scenario.

7. Related Work

QoS mechanism integration within Grid Computing environments is an interesting topic due to the complexity that it requires. There is no definitive or final solution for this problem, and research has focused on the improvement of specific QoS mechanisms.

We have structured the related work into the following parts. First, we present the existing efforts within the SLA topic. Second, we discuss the existing QoS mechanisms based on scheduling in advance and resource behavior prediction on Grid resources. Right after, a brief overview of alternative QoS

differentiation projects are outlined. And finally, the conclusions that can be extracted in order to distinguish and empower our work.

7.1. Service Level Agreements

Focusing on specific QoS mechanisms, there have been many efforts within the SLA topic, but the Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) of the Open Grid Forum (OGF) effort on SLAs standardization, resulting on the WS-Agreement specification [12] and on its extension [13], can be considered as the most important one nowadays.

It has not been the only effort on this direction, for instance, the WSLA [25] and SLAng [26] specifications appeared previously. They both are currently unmaintained, which means that WS-Agreement is nowadays the only supported SLA specification and has been widely adopted by most Grid projects interested in SLAs [27, 28, 29].

WS-Agreement specification provides the structure and negotiation protocol for SLA establishment [12]. It proposes a simple negotiation protocol which has been later extended [13] by introducing the concept of renegotiation.

There is not many information related to the SLA evaluation in terms of QoS provided by the projects. They focus on the implementation of WS-Agreement but as it is an open specification that leaves a big margin of decision/policies implementation, projects protect their advances by not providing many details.

One of the projects that has shown some efforts on SLA term evaluation is AssessGrid (*Advanced Risk Assessment and Management for Trustable Grids*) [30]. AssessGrid is focused on risk assessment in Grid environments by using SLAs. This project is very important as it introduced the concept of *negotiation broker*, whose main objective is to automate the interaction between users and service providers or between service providers in order to achieve better agreements [31].

Another interesting project dealing with SLAs is SLA@SOI (Service Level Agreement at Service Oriented Architectures) [28]. Its main objective is to empower the implantation of SLAs in Service Oriented Infrastructures (SOIs). In this project, the problem of SLA translation from observable metrics to configurable parameters is tackled [32] and the result is a framework that is able to distinguish many translation types in Service Oriented Architectures (SOAs). Nowadays, SLA@SOI is pushing to migrate its development to Cloud computing environments.

Moreover, there are proposals that contribute to the topic by proposing the use of SLAs within service-oriented Grid environments in order to provide on-demand QoS [33, 34]. Although these proposals consider the use of resource reservation as in our work, they both lack of using multiple QoS scheduling mechanisms, such as Scheduling in Advance in conjunction with resource reservation, and consequently consider a simple QoS scheme without level differentiation. Besides, the proposal presented in [33] is not WS-Agreement compliant.

Other proposals [35, 36] also consider the use of time-constrained SLAs within Grid computing in order to improve the QoS, but from different points of view. In the first proposal, the authors propose to use real-time application reconfiguration in order to achieve the highest QoS level by monitoring the performance and SLA requirements. This is a different approach to our proposal, which is based on SLA fulfillment by performing smart scheduling for each different QoS level. In the second proposal, authors describe a grid resource management mechanism (as an extension for a specific resource administrator implementation known as RAPA) focused on making optimal policies for resource selection based on system status and multiple specific QoS factors. Unluckily, this proposal has only been simulated. A more recent publication [37] considers the use of time-constrained SLAs within Grids in order to improve the negotiation strategy by using a Bayesian learning agent.

Other fields like SLA management [38, 39], QoS implications [40] or semantic and virtualization exploitation [41] have also been actively researched.

7.2. QoS Mechanisms

Focusing on QoS mechanisms, resource reservation is one of the most commonly used techniques. Projects like GARA [42] or PBS [14] (or its open source version known as Torque [15]) have aimed at providing a framework to let resource reservation for a wide variety of resources (such as CPU, main memory, network, etc.). There are plenty of articles based on those reservations that try to ensure resource availability when needed, such as [43, 44, 45]. Unluckily, Grid computing limits its functionality because not all resources can be reserved, as these mechanisms may not be available (this feature is provided by the LRMS), specially when dealing with volunteer resources. There has also been a wide discussion on the cost of reservation [17].

Meta-scheduling has also been actively researched as an alternative to reservations in advance when those are not feasible. In [46] a distributed meta-scheduling architecture that employs P2P search algorithms and a

multi-criteria decision-making is described, whilst in [23] a framework to perform meta-scheduling in advance (with rescheduling capability) is presented. However, they are not aimed at provisioning different QoS level to the users but just scheduling the jobs trying to fulfill their deadlines.

Another trend to try to ensure QoS agreements within Grid environments is by rescheduling or co-allocating jobs. Rescheduling has excelled at improving the performance of scheduling by reorganizing the schedule in order to optimize the resources usage and allow a higher allocation [23], as scheduling becomes less efficient with time. One related work focused on job reallocation is [9], where resources are provisioned along multiple sites by using backfilling techniques. In addition to that, in [47] task reallocation in Grids is analyzed through simulation, presenting and studying different reallocation algorithms and their behaviors within multi-cluster Grid environments – but only over dedicated Grid environments. Again, unlike our work, they do not provide QoS differentiation.

On the other hand, Grid workloads may be of different types (e.g., bags of tasks, workflows, pilots, etc.). There are therefore reallocation/rescheduling techniques applied to them [48]. For instance, [49] or [50] are centered on workflow rescheduling. The former performs task rescheduling following a *stop and restart* technique – not always feasible in all the scenarios. The latter tries to make the workflows makespan shorter by changing scheduling decisions over time. However, as opposite to our work, neither works take into account job time constraints when (re)scheduling jobs. As presented in [51], rescheduling techniques can be used to scheduling workflow application in a deadline-based strategy. In that work, by means of simulation, they propose a similar approach to ours, trying to both increase the utilization and meet job deadlines, but without providing any differentiation of QoS levels. These rescheduling techniques can also be used to try to provide fault tolerance performance, such as the work presented in [52], where this is provided by changing the frequency of the checkpointing process based on current status and history of failure information of the resource and by rescheduling the jobs when those failures happen. However, they do not migrate jobs to increase utilization or QoS differentiation amongst users. They only move jobs to try to keep the QoS agreed when a resource failure happens.

There have been also many efforts on Grid resource behavior prediction, as they are needed to provide an efficient scheduling. This has been actively researched in [11, 53, 54] by the study of different algorithms for resource behavior forecasting (such as exponential smoothing [55]) with interesting

results. For instance, in [55] a comparison between using or not exponential smoothing prediction methods at the scheduling decisions is presented, highlighting their impact on the performance improvement. On the contrary, forecasting is not an exact science, so it can fail and may lead to undesired situations. As a consequence, self-tuning mechanisms [18] appear as a solution in order to avoid as much as possible those unexpected situations and improve the forecasting, but introducing a general overhead. An algorithm to perform resource selection based on performance predictions, as well as though co-allocation of multiple resources is presented in [56] where already made reservations are displaced based on making co-allocation of jobs. But, unlike our work, those reschedulings are not carried out among jobs belonging to different users.

In that topic, there are also studies about how important is to have accurate run time estimations and how to handle their inaccuracy when executing BoT applications in multi-providers environments. This is evaluated in [57] where a coordinated BoT rescheduling algorithm that deals with the estimated completion time of the entire BoT application is presented.

7.3. *QoS Differentiation*

A number of mechanisms for fairshare-based job prioritization exist. In [58], the concept of resource allocation fairness to user level in uni-processor environments is extended. In [59], the Fair Execution Time Estimation (FETE) scheduling is presented, which is based on completion time predictions and focused on minimizing risk for missed job deadlines. Finally, in [19] a Grid Fair-share Scheduling system is proposed (Aequus, before known as FSGrid) that aims to provide distributed fair balancing of resource utilization between Virtual Organizations in Grid environments. Aequus performs fair-share load balancing by influencing the order in which jobs are scheduled for execution. This is done by comparing policy trees to usage trees, which inherit structure from policies and are populated with all resource usage data. Moreover, as shown in [60], this system can be easily used into the SA-Layer and extensively to the SLA management framework presented here. This makes easy to provide not only QoS differentiation among jobs but also among the different users. We can then assure that the same amount of Gold jobs are going to be accepted for each user or that some users have a greater priority and more jobs of the higher QoS have to be scheduled for them.

Alternatively, some proposals push for the QoS differentiation in terms of internal metrics related to the Grid infrastructure defined on SLAs [36, 37]

(e.g., CPU architecture, amount of RAM, bandwidth, etc.). Our work encompasses this terms by defining a high level term that represents the confidence that the users expect from the service. Thus, our proposal considers a clearly defined multi-level QoS service that can be extended in order to support new internal metrics and scheduling policies.

7.4. Conclusions

Due to Grid nature, it is possible to find a wide variety of different QoS mechanisms spread over a number of different resources within Grid environments. The most interesting and promising mechanisms have been analyzed in previous Subsections and have demonstrated to provide QoS in various ways. These mechanisms have shown to be not flexible enough in order to ensure or work within all the resources that belong to a Grid environment, even though they can be available in some of them.

Consequently, we propose an orchestrated use of the available QoS mechanisms, with the main objective of empowering the usage and fulfill the users expectations in terms of confidence. To the best of our knowledge, there is no other framework that is aimed at integrating different QoS-support mechanisms and harnessing them to provide a QoS-level based differentiation over different kind of resources that present different levels of confidence

This fact distinguishes our work and make it unique within the matter.

8. Conclusions and Future Work

Providing or guaranteeing Quality of Service within Grid environments is a very difficult and complex task, specially when volunteer resources are involved. Their distributed and heterogeneous nature, together with the variety of resources that can be available, the user expectations, the lack of total control over the resources and the amount of different QoS mechanisms that can be available, difficult the optimal exploitation of all resources.

The objective of this work is to empower a novel integrated QoS support within Grid Computing environments through SLAs by exploiting different available scheduling mechanisms in a coordinated way, improving and/or ensuring the agreed QoS through SLAs, transferring an essentially best effort architecture into a trustable infrastructure where QoS can be assured if needed. Moreover, resource usage optimization is targeted by applying the most appropriate technique to support every desired QoS level of performance.

This work presents an extension of the framework proposed previously by the authors [6], where rescheduling techniques have been included in order to improve the Scheduling and Reservation in Advance mechanisms, with the aim of further increase resource usage, QoS provisioning and minimizing the SLA violation rate. Additionally, this proposal has been evaluated in a real Grid environment testbed.

This work reveals the optimization of the resources usage by integrating and using in a combined way SLAs, Scheduling in Advance, Reservations and Rescheduling, which has an impact on providing a better QoS (guaranteed by the SLAs), improving the amount of jobs successfully processed, and enhancing the overall Grid environment productivity.

The first conclusion that must be highlighted is the validation of the proposal over the Grid environment. All mechanisms have been deployed and their integrated use comes possible with this proposal. Moreover, an improvement of the performance and behavior is obtained while keeping the priorities established for every QoS level explicitly defined by users on the SLAs.

The experimental results obtained show that the inclusion of the rescheduling mechanism avoids a big amount of SLA violations, and helps to reduce the rejection rate. Thus, rescheduling has a positive impact on the amount of highest QoS level acceptance rate while not compromising their integrity (keeping their violation rate at zero). Furthermore, the best effort jobs waiting time is considerably reduced.

Some future work guidelines are: 1) Improve the SLA manager with the WS-Agreement extension, 2) include new metrics, and 3) extend the proposal with more QoS mechanisms. Also, the migration of the proposal and framework to Cloud computing environments will be addressed in the near future.

Acknowledgements

This work was supported by the Spanish Government under Grant TIN2012-38341-C04-04 and through a FPI scholarship associated to TIN2009-14475-C04-03 project.

References

- [1] D. Armstrong, K. Djemame, Towards Quality of Service in the Cloud, in: Proc. of the 25th UK Performance Engineering Workshop, Leeds, UK., 2009.
- [2] V. Stantchev, C. Schröpfer, Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing, in: Proc. of the 4th Intl. Conference on Advances in Grid and Pervasive Computing (GPC), Geneva, Switzerland, 2009.
- [3] J. Padgett, K. Djemame, P. Dew, Grid-Based SLA Management, in: Proc. of the European Grid Conference (EGC), Amsterdam, The Netherlands, 2005.
- [4] E. Huedo, R. Montero, I. Llorente, A Modular Meta-scheduling Architecture for Interfacing with pre-WS and WS Grid Resource Management Services, *Future Generation Computer Systems* 23 (2) (2007) 252–261.
- [5] GWD-I, Advance Reservations: State of the Art, Tech. rep., Global Grid Forum (GGF) (June 2003).
- [6] J. Conejero, L. Tomás, C. Carrión, B. Caminero, Multilevel SLA-based QoS support in Grids, in: 10th IEEE Intl. Symposium on Parallel and Distributed Processing with Applications (ISPA), Leganés. Spain, 2012.
- [7] J. Conejero, C. Carrión, B. Caminero, A SLA-based Framework with Support for Meta-scheduling in Advance for Grids, in: Proc. of the Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA, 2011.
- [8] I. T. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems., in: H. Jin, D. A. Reed, W. Jiang (Eds.), NPC, Vol. 3779 of Lecture Notes in Computer Science, Springer, 2005, pp. 2–13.
- [9] M. D. De Assunção, R. Buyya, Performance Analysis of Multiple Site Resource Provisioning: Effects of the Precision of Availability Information, in: Proc. of the 15th Intl. Conference on High Performance Computing (HiPC'08), Springer-Verlag, Berlin, Heidelberg, 2008, pp. 157–168.

- [10] P. R. Wilson, M. S. Johnstone, M. Neely, D. Boles, Dynamic Storage Allocation: A Survey and Critical Review, in: Proceedings of the International Workshop on Memory Management, IWMM '95, Springer-Verlag, London, UK, UK, 1995, pp. 1–116.
- [11] A. Caminero, O. Rana, B. Caminero, C. Carrión, Performance Evaluation of an Autonomic Network-aware Metascheduler for Grids, Concurrency and Computation: Practice and Experience 21 (13) (2009) 1692–1708.
- [12] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web Services Agreement Specification (WS-Agreement) GFD-R-P.192, Tech. rep., <https://forge.gridforum.org/projects/graap-wg/> (October 2011).
- [13] O. Waelrich, D. Battrè, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, W. Ziegler, WS-Agreement Negotiation Version 1.0 GFD-R-P.193, Tech. rep., <https://forge.gridforum.org/projects/graap-wg/> (October 2011).
- [14] PBS Works - Enabling On-Demand Computing, Web page at <http://www.pbsworks.com/> (Last access: 4th November, 2012).
- [15] TORQUE Resource Manager, Web page at <http://www.adaptivecomputing.com/products/open-source/torque/> (Last access: 4th November, 2012).
- [16] G. Borges, M. David, J. Gomes, C. Fernandez, J. Lopez Cacheiro, P. Rey Mayo, A. Simon Garcia, D. Kant, K. Sephton, Sun Grid Engine, a new Scheduler for EGEE Middleware, in: Proc. of the IBERGRID - Iberian Grid Infrastructure Conference, 2007.
- [17] C. Zhao, J. Cao, H. Wu, F. Zhang, Cost Estimation of Advance Reservations over Queued Jobs: a Quantitative Study, Intl. Journal of Modeling, Simulation, and Scientific Computing 1 (3) (2010) 317–332.
- [18] L. Tomás, A. C. Caminero, C. Carrión, B. Caminero, Network-aware Meta-scheduling in Advance with Autonomous Self-tuning System, Future Generation Computer Systems 27 (5) (2011) 486 – 497.

- [19] P.-O. Östberg, D. Henriksson, E. Elmroth, Decentralized, Scalable, Grid Fairshare Scheduling (FSGrid), *Future Generation Computer Systems* 29 (1) (2013) 130–143.
- [20] C. Vázquez, E. Huedo, R. S. Montero, I. M. Llorente, Federation of TeraGrid, EGEE and OSG Infrastructures through a Metascheduler, *Future Generation Computing Systems* 26 (2010) 979–985.
- [21] Torque Resource Manager, Web page at <http://www.adaptivecomputing.com/products/torque.php> (Last access: 27th September, 2012).
- [22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd Edition, The MIT Press, 2001.
- [23] L. Tomás, B. Caminero, C. Carrión, A. C. Caminero, On the Improvement of Grid Resource Utilization: Preventive and Reactive Rescheduling Approaches, *Journal of Grid Computing* 10 (3) (2012) 475–499.
- [24] G. Chun, H. Dail, H. Casanova, A. Snaveley, Benchmark Probes for Grid Assessment, in: *Proc. of the Intl. Parallel & Distributed Processing Symposium (IPDPS)*, Santa Fe, USA, 2004.
- [25] WSLA: Web Service Level Agreements, Web page at <http://www.research.ibm.com/wsla/> (Last access: 20th October, 2012).
- [26] D. D. Lamanna, J. Skene, W. Emmerich, SLaNg: A Language for Defining Service Level Agreements, in: *Proc. of the Intl. Workshop of Future Trends of Distributed Computing Systems*, San Juan, Puerto Rico, 2003.
- [27] D-Grid - SLA4D-Grid - Service Level Agreement für das D-Grid, Web page at <http://www.d-grid-ggmbh.de/index.php?id=89> (Last access: 21th October, 2012).
- [28] SLA at SOI, Web page at <http://sla-at-soi.eu/> (Last access: 21th October, 2012).
- [29] WSAG4J - WS-Agreement framework for Java, Web page at <http://packcs-e0.scai.fraunhofer.de/wsag4j/> (Last access: 20th October, 2012).

- [30] AssessGrid, Web page at <http://www.assessgrid.eu> (Last access: 24th March, 2012).
- [31] M. Parkin, R. M. Badia, J. Martrat, A Comparison of SLA Use in Six of the European Commissions FP6 Projects, Tech. Rep. TR-0129, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence (April 2008).
- [32] H. Li, W. Theilmann, J. Happe, Challenges in SLA Translation, SLA@SOI Technical Paper <http://sla-at-soi.eu/results/publications/>.
- [33] G. Engelbrecht, S. Benkner, A Service-Oriented Grid Environment with On-demand QoS Support, in: SERVICES I, IEEE Computer Society, 2009, pp. 147–150.
- [34] J. Padgett, K. Djemame, P. Dew, Grid Service Level Agreements Combining Resource Reservation and Predictive Run-time Adaptation, in: UK e-Science All Hands Meeting, Nottingham UK, 2005.
- [35] A. Menychtas, D. Kyriazis, K. Tserpes, Real-time Reconfiguration for Guaranteeing QoS Provisioning Levels in Grid Environments, *Future Generation Computing Systems* 25 (7) (2009) 779–784.
- [36] Y. Han, C.-H. Youn, A new Grid Resource Management Mechanism with Resource-aware Policy Administrator for SLA-constrained Applications, *Future Generation Computing Systems* 25 (7) (2009) 768–778.
- [37] G. C. Silaghi, L. D. Șerban, C. M. Litan, A time-constrained SLA Negotiation Strategy in Competitive Computational Grids, *Future Generation Computing Systems* 28 (8) (2012) 1303–1315.
- [38] J. Padgett, K. Djemame, P. Dew, Predictive Adaptation for Service Level Agreements on the Grid, *Intl. Journal of Simulation Systems, Science and Technology* 7 (2) (2006) 29–42.
- [39] W. Theilmann, L. Baresi, Multi-level SLAs for Harmonized Management in the Future Internet, IOS Press, 2009, Ch. Towards the Future Internet, pp. 193–202. doi:10.3233/978-1-60750-007-0-193.

- [40] I. Brandic, D. Music, S. Dustdar, S. Venugopal, R. Buyya, Advanced QoS Methods for Grid Workflows Based on Meta-Negotiations and SLA-Mappings, in: Proc. of the 3rd Workshop on Workflows in Support of Large-Scale Science. In conjunction with Supercomputing, Austin, USA, 2008.
- [41] J. Ejarque, M. de Palol, I. Goiri, F. Julià, J. Guitart, R. M. Badia, J. Torres, Exploiting Semantics and Virtualization for SLA-driven Resource Allocation in Service Providers, *Concurrency and Computation: Practice and Experience* 22 (5) (2010) 541–572.
- [42] A. Roy, V. Sander, Grid resource management, Kluwer Academic Publishers, Norwell, MA, USA, 2004, Ch. GARA: a uniform quality of service architecture, pp. 377–394.
- [43] E. Elmroth, J. Tordsson, Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions, *Future Generation Computing Systems* 24 (6) (2008) 585–593.
- [44] C. Qu, 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.
- [45] A. Sulistio, Advance Reservation and Revenue-based Resource Management for Grid Systems, Ph.D. thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia (May 2008).
- [46] J. Heilgeist, T. Soddemann, H. Richter, Distributed Meta-scheduling for Grids, in: Proc. of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities, MG '08, ACM, New York, NY, USA, 2008, pp. 25:1–25:1.
- [47] Y. Caniou, G. Charrier, F. Desprez, Analysis of Tasks Reallocation in a Dedicated Grid Environment, in: Proc. of the Intl. Conference on Cluster Computing (CLUSTER), Heraklion, Greece, 2010.

- [48] A. Iosup, D. Epema, Grid Computing Workloads, *Internet Computing*, IEEE 15 (2) (2011) 19–26.
- [49] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, J. Dongarra, New Grid Scheduling and Rescheduling Methods in the GrADS Project, in: *Proc. of the 18th Intl. Parallel and Distributed Processing Symposium*, Santa Fe, USA, 2004.
- [50] Z. Yu, W. Shi, An Adaptive Rescheduling Strategy for Grid Workflow Applications, in: *Proc. of the 21th Intl. Parallel and Distributed Processing Symposium*, Long Beach, USA, 2007.
- [51] W. Chen, A. Fekete, Y. C. Lee, Exploiting Deadline Flexibility in Grid Workflow Rescheduling, in: *Proc. of the 11th IEEE/ACM Intl. Conference on Grid Computing*, Brussels, Belgium, 2010.
- [52] S. A. L. Therasa, G. Sumathi, S. A. Dalya, Article: Dynamic Adaptation of Checkpoints and Rescheduling in Grid Computing, *International Journal of Computer Applications* 2 (3) (2010) 95–99.
- [53] M. Kalantari, M. K. Akbari, Grid Performance Prediction using State-space Model, *Concurrency and Computation: Practice and Experience* 21 (9) (2009) 1109–1130.
- [54] H. Jin, X. Shi, W. Qiang, D. Zou, An Adaptive Meta-scheduler for Data Intensive Applications, *Intl. Journal of Grid and Utility Computing* 1 (1) (2005) 32–37.
- [55] L. Tomás, A. Caminero, C. Carrión, B. Caminero, Exponential Smoothing for Network-aware Meta-scheduler in Advance in Grids, in: *Proc. of the 6th Intl. Workshop on Scheduling and Resource Management on Parallel and Distributed Systems (SRMPDS)*, San Diego, USA, 2010.
- [56] E. Elmroth, J. Tordsson, A Standards-based Grid Resource Brokering Service Supporting Advance Reservations, Coallocation, and Cross-Grid Interoperability, *Concurrency and Computation: Practice and Experience* 21 (18) (2009) 2298–2335.

- [57] M. A. Netto, R. Buyya, Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers, *Concurrency and Computation: Practice & Experience* 24 (12) (2012) 1362–1376.
- [58] J. Kay, P. Lauder, A Fair Share scheduler, *Commun. ACM* 31 (1) (1988) 44–55.
- [59] E. Dafouli, P. Kokkinos, E. A. Varvarigos, Fair Execution Time Estimation Scheduling in Computational Grids, in: P. Kacsuk, R. Lovas, Z. Nmeth (Eds.), *Distributed and Parallel Systems*, Springer US, 2008, pp. 93–104.
- [60] L. Tomás, P. Östberg, B. Caminero, C. Carrión, E. Elmroth, Addressing QoS in Grids through a Fairshare Meta-scheduling In-Advance Architecture, in: *Proc. Seventh Intl. Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Victoria, Canada, 2012.