Computing and Informatics, Vol. 31, 2012, 73–88

QOS PROVISIONING BY META-SCHEDULING IN ADVANCE WITHIN SLA-BASED GRID ENVIRONMENTS

Javier Conejero, Luis Tomás
Blanca Caminero, Carmen Carrión

Department of Computing Systems
University of Castilla-La Mancha
02071, Albacete, Spain
e-mail: {FJavier.Conejero, Luis.Tomas, MariaBlanca.Caminero, Carmen.Carrion}@uclm.es

Abstract. The establishment of agreements between users and the entities which manage the Grid resources is still a challenging task. On the one hand, an entity in charge of dealing with the communication with the users is needed, with the aim of signing resource usage contracts and also implementing some renegotiation techniques, among others. On the other hand, some mechanisms should be implemented which decide if the QoS requested could be achieved and, in such case, ensuring that the QoS agreement is provided. One way of increasing the probability of achieving the agreed QoS is by performing meta-scheduling of jobs in advance, that is, jobs are scheduled some time before they are actually executed. In this way, it becomes more likely that the appropriate resources are available to run the jobs when needed. So, this paper presents a framework built on top of Globus and the GridWay meta-scheduler to provide QoS by means of performing meta-scheduling in advance. Thanks to this, QoS requirements of jobs are met (i.e. jobs are finished within a deadline). Apart from that, the mechanisms needed to manage the communication between the users and the system are presented and implemented through SLA contracts based on the WS-Agreement specification.

Keywords: Grid meta-scheduling, QoS, SLAs, WS-Agreement

1 INTRODUCTION

In highly variable and heterogeneous systems, like Grid environments, where resources may be scattered across multiple domains and under different access policies, it is extremely difficult to provide QoS to users. Hence, the Grid infrastructure must provide the needed services for automatic resource brokerage which take care of the resource selection and negotiation process [1]. This infrastructure is named "*meta-scheduler*" [2] and hides this process from the user. However, the scheduling process is complicated due to several facts, like heterogeneous and distributed nature of resources, different characteristics of the applications, and specially due to the fact that the meta-scheduler entity typically lacks total control and even complete knowledge of the system resources. This means that it is not always possible to reserve the resources selected for executing the jobs which finally implies that it is not possible to ensure the execution of a job into a resource within the requested time.

Thus, some mechanisms to provide QoS in such kind of environments have to be developed. As reservations in advance are not always feasible, the idea is to try to ensure that a specific resource is available when a job requires it through meta-scheduling in advance of resources. This meta-scheduling in advance algorithm can be defined as the first step of the reservations in advance algorithm, in which the resources and the time periods to execute the jobs are selected (and the system keeps track of the decisions already made and the usage of resources) but making no physical reservation. This way, the system needs to estimate the future resource status (when the jobs to schedule will be executed) and how long their execution will be. To this end, some predictions techniques are implemented.

On the other hand, and taking into consideration that the user's experience of the Grid is determined by the functionality and performance of this meta-scheduler system, it is needed to develop some mechanism to deal with the interaction among users and the meta-scheduling system. To this end, Service Level Agreements (SLAs [3]) contracts are the main mechanism to achieve this goal.

Nowadays economy is moving from product oriented economy to a service oriented economy in computational environments. This trend requires new mechanisms to manage and enforce the use of computational resources in an efficient way for an optimized exploitation of them; but this exploitation is strongly managed by economical and business motivations, where mechanisms to enforce and negotiate legal statements are needed [4]. Due to this fact, many efforts have been done to tackle this problem within Grid environments, resulting in the *Grid Resource Allocation Agreement Protocol* (WS-Agreement) specification [5] for SLAs. Thenceforth, there are many Grid projects interested in the implantation and use of SLAs (e.g. AssessGrid [6], Brein [7], and SLA@SOI [8]).

For our purpose, SLAs represent a formalization of the job submission process for the Grid. Furthermore, they are a mechanism for a formal representation of the temporal restrictions that correspond to the associated job, which are used in the meta-scheduling in advance process, with the objective of a QoS improvement.

To sum up, the main contribution of this paper is a *framework* built on top of the GridWay meta-scheduler [9] to provide QoS by means of performing meta-scheduling in advance, with a SLA-based user interface. The usage of this framework allows jobs to be executed within their deadlines thanks to some implemented heuristics which estimate the future status of resources and how long a job execution will be.

The paper is organized as follows. Related work is presented in Section 2. In Section 3 the framework to perform SLA-based meta-scheduling in advance is outlined. Section 4 shows the methodology to carry out the communication process between the users and the system through SLA contracts. Finally, the conclusions obtained and the suggested guidelines for future work are outlined in Section 5.

2 RELATED WORK

The provision of QoS in Grids is still an open issue which has been explored by several research projects, based on advanced reservation, such as *GARA* [10], *Grid Capacity Planning* [11], or *VIOLA* [12], among others. All these techniques have the same main drawback, namely not all the resources can be reserved for several reasons (e.g. not all the resources provide this functionality). Due to this limitation, our work aims at performing scheduling in advance rather than reservations of resources in advance.

Meta-scheduling in advance needs to perform predictions about the future resource status and about job duration into resources. A survey of some prediction techniques can be found in [13]. Examples include applying statistical models to previous executions [14] and heuristics based on job and resource characteristics [15]. In [14], it is shown that although load exhibits complex properties, it is still consistently predictable from past behaviour. In [15], an evaluation of various linear time series models for prediction of CPU loads in the future is presented. In our work, a technique based on historical data is used, since it has been demonstrated to provide better results compared to linear functions [16].

This kind of scheduling needs to have a suitable data structure to be able to manage all the information efficiently. There are several structures for managing this information needed by the scheduler, for instance Grid Advanced Reservation Queue [17] (GarQ), and a survey can be found in [17]; but in this work, *red–black trees* are used since they provide us with efficient access to the information about resource usage, as it has been demonstrated in [18].

On the other hand, SLAs are a hot topic nowadays. Many efforts have been done in several fields, like their management [19], QoS implications [20], semantic and virtualization exploitation [21] and especially in their standardization. The most important improvement within SLAs has been the WS-Agreement specification [5], which is considered the "de-facto" standard. The structure and mechanisms to deploy SLAs over a system are described from a global point of view, and thanks to the recent revision of the WS-Agreement specification [22], a new negotiation protocol has been defined, introducing the renegotiation concept as a multiple message in-

teraction between user and service provider to achieve better agreements. However, WS-Agreement is not the only available specification. SLAng [23] and WSLA [24] are alternatives to it, but due to their lack of support they are not recommended.

Due to the Service Level Agreements importance, many projects are interested on its implementation [25]. Most of them implement the WS-Agreement, like SLA@SOI [8], AssessGrid [6] and Brein [7]. The first one is focused on the implantation of SLAs into Service Oriented Infrastructures (SOIs [8]) from a generic point of view. AssessGrid and Brein have a common purpose, which is to promote Grid computational environments into business environments and society. However, AssessGrid is focused on risk assessment for trustable Grids while Brein is focused on an efficient handling and management of Grid computing based on artificial intelligence, web semantics and intelligent systems. Another important project within this matter is WSAG4J(*WS-AGreement for Java* [26]), which is a generic implementation of the WS-Agreement specification developed by the Fraunhofer SCAI Institute as a development framework. It is designed for a quick development and debugging of services and applications based on WS-Agreement.

It should be noted that not all projects implement WS-Agreement for their SLA management. An example is represented by NextGrid [27], which is focused on business Grid exploitation.

3 SCHEDULING IN ADVANCE FRAMEWORK (SA-LAYER)

In a real Grid environment, many resources cannot be reserved, due to the fact that not all the local resource management systems permit them. Apart from that, there are other types of resources such as bandwidth, which are shared among several administrative domains making their reservation more difficult or even impossible. This is the reason to perform meta-scheduling in advance rather than reservations in advance to provide QoS in Grids. This means that the system keeps track of the meta-scheduling decisions already made to take future decisions and with the aim of not overlapping executions. However, no physical reservations are done. So, our *scheduling in advance process* follows the next steps (see Figure 1):

1. A user sends a request to the meta-scheduler at his local administrative domain through the SLA manager (see Section 4). Every SLA contract (job execution request) must provide a tuple with information on the application and the input QoS parameters: (in_file, app, t_s, d). in_file stands for the input files required to execute the application, *app*. In this approach the input QoS parameters are just specified by the start time, t_s (earliest time jobs can start to be executed), and the deadline, d (time by which jobs must have been executed).
2. The meta-scheduler communicates with the *Gap Management* entity to obtain both the resource and the time interval to be assigned for the execution of the job. The heuristics algorithms presented here take into account the predicted state of the resource (both for computational resources and interconnection networks), the jobs that have already been scheduled and the QoS requirements of the job.

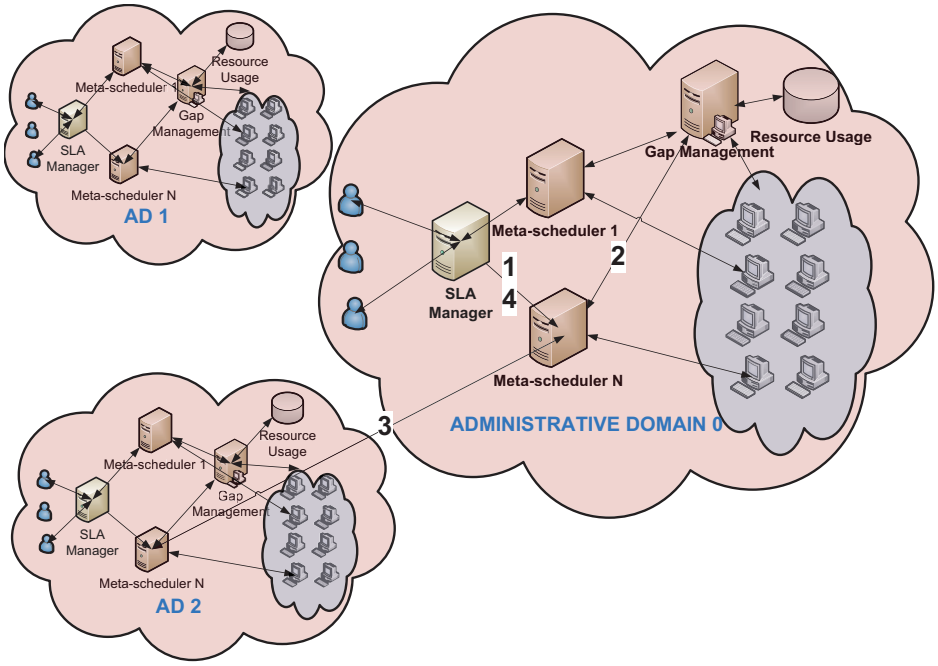

Fig. 1. Meta-Scheduling in advance process

3. If it is not possible to fulfill the user's QoS requirements using the resources of its own domain, a communication with meta-scheduler of other domains starts. In order to perform the inter-domain communications efficiently, techniques based on P2P systems (as proposed by [28, 29], among others) can be used. This way, the meta-scheduler at each domain knows some of the meta-schedulers at other domains, and can forward jobs to them when necessary.
4. If it is still not possible to fulfill the QoS requirements (not even in other domains), a renegotiation process is started between the user and the SLA manager in order to define achievable QoS requirements. Recall that this renegotiation, as well as the overall interaction with users, is conducted by means of Service Level Agreements (SLAs). A scheme for advancing and managing QoS attributes contained in Grid SLAs contracts is implemented and detailed in Section 4.

As Figure 1 depicts, there may be more than one meta-schedulers in each local administrative domain (subdomains of a Virtual Organization (VO)), albeit they have to communicate with the same Gap Management entity. The Gap Management entity has the information about future usage of the resources of its domain and could also be replicated to avoid the single point of failure problem. Even the resources may be split into several subdomains in case of a huge number of them, making it quite scalable. This represents an idealistic scenario where all the jobs are submitted

through the Gap Management entities in charge of the resources usage. However, this is not the rule into a real Grid environment, where resources usually are shared among users and VOs. For this reason, the system needs to estimate the future resources status for taking into account the resources load which is not submitted through the meta-scheduling in advance process. So, all the load not submitted through our system, as resource owners' load or the rest of jobs submitted by using other meta-schedulers of the VO, is considered as a load into the resource that must be predicted.

This meta-scheduling in advance functionality has been implemented as a layer on top of the GridWay meta-scheduler [2], called *Scheduler in Advance Layer (SA-layer) [16]*, as Figure 2 depicts. The *SA-layer* uses functionality provided by GridWay in terms of resource discovery and monitoring, job submission and execution monitoring, etc.. Also, the information concerning previous jobs executions and the status of resources and network over time are stored in *DB Executions* and *DB Resources*, respectively.

The usage of the resources is divided into time intervals, named *slots*. So, the system has to schedule the future usage of resources by allocating the jobs into the resources at one specific time (taking one or more time slots). Therefore, data structures (represented by *Data Structure* in Figure 2) to keep a trace of the slots usage are needed. In this work the *red-black trees* [18] are used as a data structure with the objective of developing techniques to efficiently identify feasible idle periods, without having to examine all idle periods. The reason for choosing this kind of structure is its property which enforce that the longest path from the root to any leaf is no more than twice as long as the shortest path from the root to any other leaf in that tree. So, the tree is roughly balanced, and as a result of that, inserting, deleting and finding values require worst-case time proportional to the height of the tree ($O(\log n)$). The idea of using red-black trees was firstly proposed by Castillo et al. [18]. Nevertheless, their proposal does not take into account the performance fluctuation. Moreover, authors of [18] assume that users have prior knowledge on the duration of jobs, which is not necessarily true in a real grid. Our work does not depend on such assumption, so there is a necessity of developing algorithms for estimating job durations into resources (*Predictor* in Figure 2), and consequently, to infer how many slots a job will need to be executed in a certain resource.

3.1 Job Completion Time Predictions

The different performance of Grid resources makes rather difficult to obtain predictions about jobs durations into resources. What is more, the job performance characteristics may vary for different applications and from time to time. Due to these facts, it is needed to estimate the future status of resources and taking it into account, estimating the time needed to complete the job in a resource at the target time interval. With the objective of making those predictions as accurate as possible, they are calculated by estimating the execution time of the job and the time needed to complete the transfers separately. To do that, the system takes

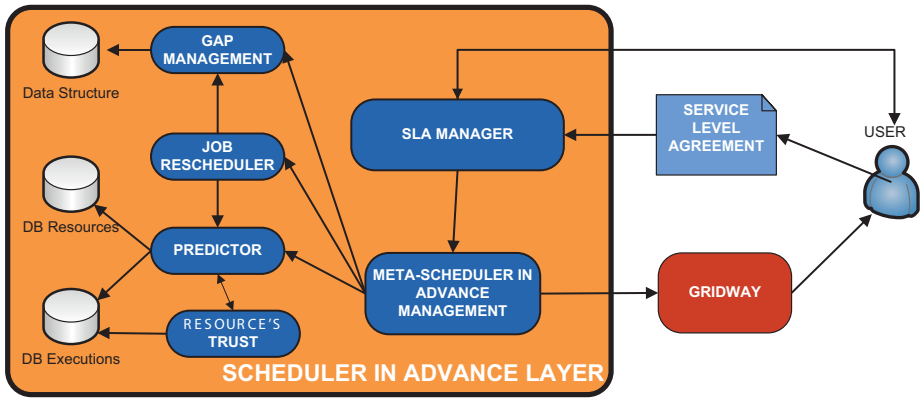

Fig. 2. The Scheduler in advance layer (SA-layer)

into account the characteristics of the jobs, the power and usage of the CPU of the resources and the network status. To this end, our system implements a technique based on an Exponential Smoothing function which calculates the future status of the resource CPUs and the future status of the network links. For more information about this function see [30].

Taking into account the information about Grid status, an estimation of the execution time is calculated using information of previous executions, as depicted in Algorithm 1. This algorithm uses all the execution times records in the database (which are stored in *DB_Executions*) for the application *app* in a resource R_i to calculate the mean execution time for *app* in R_i – this includes execution and queueing times (line 8). After that, the prediction on the future CPU status of each resource is calculated by means of an exponential smoothing function (line 9). Finally, the mean execution time is tuned by using the prediction about the future CPU status of each resource (line 10). The way of calculating the transfer times is pretty similar. The mean bandwidth predicted for the time period between the job start time and its deadline is calculated through an exponential smoothing function. Then, using this information along with the total number of bytes to transfer, the time needed to complete the transfers is estimated.

Finally, the predictions obtained are weighted taking into account the trust into the resources chosen. This implementation is explained in Algorithm 2. With the estimations on execution and transfer times and the information about trust in resource R_i, labelled as $RT(R_i)$, the execution time is tuned (line 12) and an estimation for the total completion time of the job, JT_{R_i}, is calculated (line 14).

The trust on resources is computed as Equation (1) denotes:

$$RT(R_i) = \frac{\sum_{j=(n-N)}^{n} \left(Estimated_{(j,i)} - Real_{(j,i)} \right)}{N} \tag{1}$$

where $Estimated_{(j,i)}$ is the job completion time estimation made for the j execution in the resource R_i; and $Real_{(j,i)}$ is the real completion time of a job j in the resource R_i. The output of this function is the mean of the errors made in those N predictions and it is used to tune the prediction made for the job execution times in that resource. As a result, the confidence in the estimations depends on how trustworthy is the resource where the job will be run. The benefits of tuning the obtained prediction by using this trust factor were evaluated in [16] highlighting the usefulness of this approach. So, now we are mixing the estimation techniques presented in [16] and [30] to obtain a more accurate prediction.

Algorithm 1 Estimation of execution time (ExecT_Estimation)

1: Let R = set of resources known to GridWay $\{R_1, R_2, \ldots, R_n\}$
2: Let *app* the job to be executed
3: Let *initT* the start time of the job
4: Let d the deadline for the job
5: Let $ExecutionTime(app, R_i)_j$ the j execution time for the application *app* in the resource R_i
6: Let $ES_{cpu}(DB\_Resources_{R_i}, initT, d)$ the exponential smoothing function that calculates the percentage of free CPU in resource R_i between time *initT* and d
7: Let $CPU\_free(R_i, initT, d)$ the percentage of free CPU in the resource R_i from time *initT* to time d
8: $ExecutionTime = \frac{\sum_{j=1}^{n} ExecutionTime(app,R_i)_j}{n}$
9: $CPU\_free(R_i, initT, d) = ES_{cpu}(DB\_Resources_{R_i}, initT, d)$
10: $ExecutionTime = ExecutionTime * (2 - CPU\_free(R_i, initT, d))$
11: return *ExecutionTime*

Algorithm 2 Job Completion Time Estimation

1: Let R_i = a resource
2: Let *app* = the job to be executed
3: Let *initT* = the start time of the job
4: Let d = the deadline for the job
5: Let $size_{IN}$ = the number of input bytes to be transferred
6: Let $size_{OUT}$ = the number of output bytes to be transferred
7: **for** each R_i having a gap **do**
8: $Prolog = TransT\_Estimation(R_i, initT, d, size_{IN})$
9: $Epilog = TransT\_Estimation(R_i, initT, d, size_{OUT})$
10: $ExecT = ExecT\_Estimation(R_i, app)$
11: **if** $RT(R_i) < 0$ **then**
12: $ExecT = ExecT + |RT(R_i)|$
13: **end if**
14: $JT_{R_i} = Prolog + ExecT + Epilog$
15: **end for**

It is important to highlight that predictions are only calculated when a suitable gap has been found in the host. In this way there is no need to calculate the completion times for all the hosts in the system – which would be quite inefficient. On the other hand, when a resource suddenly quits the system (e.g. the resource fails or it is shut down), the jobs scheduled on it (including currently running jobs) have to be reallocated to other hosts. The way how jobs are rescheduled is the same as when they were first submitted to the system. This feature is very important in Grids since resources may join and leave the Grid at any time, and failures of resources are the rule rather than the exception. This task is performed by the "*Job Rescheduler*" module (see Figure 2). Finally, the jobs that are able to manage this layer are simple jobs; but dealing with workflows and pilot jobs is about our future work.

4 SERVICE LEVEL AGREEMENTS (SLAS)

Once the execution of the job may be ensured by the system with enough accuracy, the next step to address is the communication with the user in order to reach agreements for executing his/her jobs. This process is carried out through Service Level Agreements (SLAs). The SLA concept within Grid computing is defined as a contract between user and service provider. On this contract, the expectations, obligations and legal implications are explicitly defined [3]. So, it can be said that the QoS that the user expects to receive is represented on each SLA. Furthermore, SLAs are the main mechanism to improve the commercial expansion of Grid computing due to their support for *pay-per-use* models and the fact of being a legal statement [31]. Nowadays, these points are very important because of the business interest on exploiting Grid computing.

Formally, the use of SLAs enforces the relationship between user and service provider in two ways: as a legal statement that must be accomplished and as an agreement that can be negotiated. Negotiation implies that the service provider has the opportunity to decide in advance if the user requirements can be fulfilled and if possible, to negotiate with it to reach a better agreement. Moreover, the use of SLAs improves the interoperability among Grids and from users to manage multiple Grids; but this can only be a reality if a robust and realistic standard is applied.

Nowadays, the most important and widely used standard within SLAs is the WS-Agreement. The last version of its specification was released in March 2007 [5], and with it, all the aspects related to the creation, structure and SLA management were defined. WS-Agreement defines a basic scheme for the agreements as Figure 3 illustrates. Each agreement has a *name* identifier and a *context*. In the *context*, all the information about the document is defined, like service provider information. The *terms* block consists on two subblocks: the first one, known as *service terms*, has the information relative to the services/resources that are going to be provided (e.g. CPU count, CPU architecture, RAM amount, etc.); and the second one,

known as *guarantee terms*, has the service level that must be guaranteed for each service/resource of the service terms (e.g. 2 (CPU count), x86_64 (CPU architecture), 2 GB (RAM), etc.). Finally, the *creation constraints* block is used for setting limitations on a negotiation and this block can only be defined on the template. On the negotiation process it is not used.

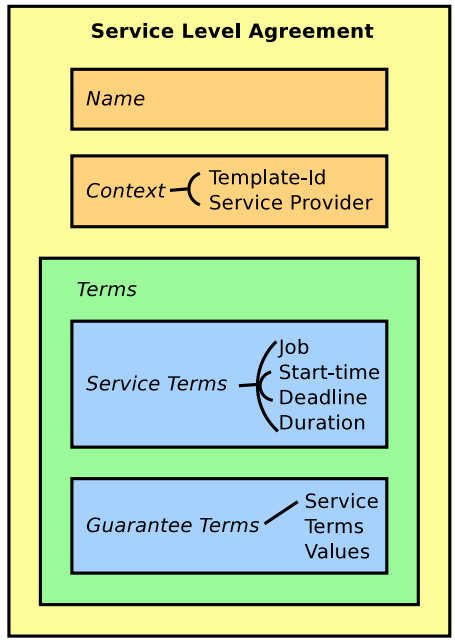

Fig. 3. SLA structure

In January 2011 a new revision of the WS-Agreement was released [22]. In this revision, an extension of the negotiation protocol defined on the first release is presented. The negotiation protocol previously defined on WS-Agreement only contemplates a simple negotiation workflow, where the user requests one template or more, fills it with the requested QoS and sends it back to the service provider, which accepts or rejects the SLA; but with the recent extension (see Figure 4), renegotiation is available through a loop between the user and service provider before an offer is committed. This allows to achieve a better agreement for both participants.

The definition of the terms in every SLA is not defined in the WS-Agreement specification, so their definition is left to the service provider, who is in charge of specifying the terms for its own needs. This flexibility of the WS-Agreement specification lets the service provider define the terms related to hardware needed, time restrictions or job related restrictions. These terms can be very numerous and different, but there are several ones that may appear: related to the hardware needed (like

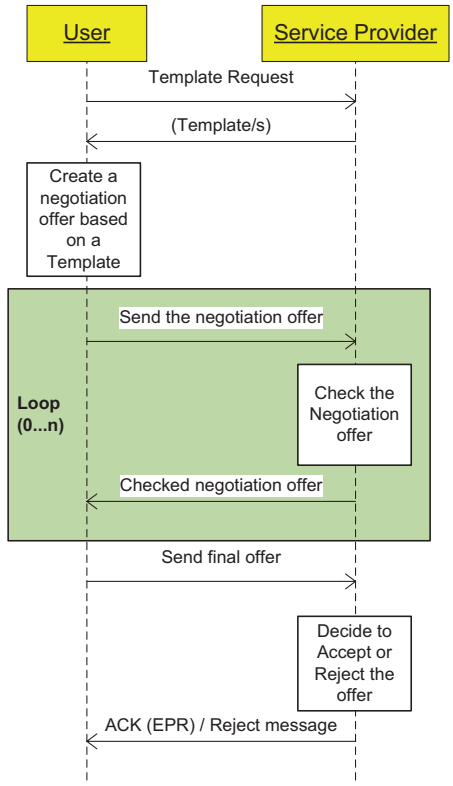

Fig. 4. WS-Agreement negotiation protocol

number of CPUs or amount of RAM among others), and more important, related to time restrictions. These restrictions often appear as start-time and duration (or deadline) of a job; but it is possible to define new terms to improve the knowledge of the jobs and to exploit them into the meta-scheduling process.

For this purpose, each SLA submitted to this framework should follow the WS-Agreement specification. So, the *service terms* specified on each SLA are the job and execution parameters needed for the meta-scheduling process (see Figure 3). These terms include: *job* (*app, in_file*), *start-time* (*t_s*) and *deadline* (*d*). The *name* block only specifies the agreement name for a better human identification while the *context* block contains two main parameters: *template-id* for internal identification and *service provider* for service provider name identification. This structure is open for future term and context parameter extension. Finally, *creation constraints* are not expected.

This framework implements the WS-Agreement specification and it is possible to interact with it through a web portal (see Figure 5). This portal offers the main fields to fill from a template. Once submited, the information is converted to

Fig. 5. SLAs web portal

an offer and sent to the SLA manager. The result of the request is shown to the user through the portal, and if the submission has been succesfull the EPR is returned. SLA monitoring and the inclusion of the negotiation extension represents the next milestones of our work.

There are several advantages that emerge from the use of SLAs into our system and more specially from the implantation of the WS-Agreement specification. First of all, it represents a formalization of the job submission process. Moreover, they are a mechanism for a formal representation of the temporal restrictions that the user sets and that our Grid system has to respect.

Finally, SLAs are XML format messages (as specified in WS-Agreement), so they can be easily handled within Web environments. Therefore, technologies such as Gridsphere [32] can be exploited for the Web environment development. This way, the complexity of the system is hidden to the user, who has the ability of interacting with the Grid through a Web portal by filling the jobs to be executed and their requirements. Then, this information is translated to a SLA and sent to the job submission process in an easy and systematic way.

5 CONCLUSIONS AND FUTURE WORK

Several research works aim at providing QoS in Grids by means of *advanced reservations*, albeit making reservations of resources is not always possible in this kind of scenario. So, this paper proposes a SLA-based framework to perform meta-scheduling in advance (first step of the reservation in advance process) to provide QoS to Grid users. Nonetheless, this type of scheduling requires to estimate whether a given application can be executed before the deadline specified by the user. So,

this requires to tackle many challenges, such as predicting the jobs completion time into the resources.

For this reason, the system is concerned with the dynamic behaviour of the Grid resources, their usage, and the characteristics of the jobs. Furthermore, this system takes into account the accuracy in the recent predictions for each resource in order to calculate a resource trust.

Moreover, a SLA manager is implemented to deal with the user interaction and to enable QoS agreements between both of them. This module manages the communication between the system, by interacting with SA-Layer and the users, and makes possible to provide QoS to the users in a contractual way (through SLAs). Furthermore, each SLA can specify more job related information that can be used in the meta-scheduling process than usual job submission.

One interesting guideline for future research is the development of techniques to perform better estimations for the transfer times. For this reason, it is a good point to try to reserve network bandwidth when and where this could be possible. Moreover, work on developing algorithms to schedule data as another resource is also considered for future research. Finally, another issue that can be addressed is the improvement of the SLA manager to make more efficient the scheduling by taking into account the associated costs, such as reducing the wasted energy.

Acknowledgements

This work was supported by the Spanish MEC and MICINN, as well as European Commission FEDER funds, under grants CSD2006-00046, TIN2009-14475-C04 and through a FPI scholarship asociated to TIN2009-14475-C04-03 project. It was also partly supported by JCCM under grant PII1C09-0101-9476.

REFERENCES

[1] Schwiegelshohn, U. et al.: Perspectives on Grid Computing. Future Generation Computer Systems, Vol. 26, 2010, No. 8, pp. 1104–1115.

[2] Huedo, E.—Montero, R. S.—Llorente, I. M.: A Modular Meta-Scheduling Architecture for Interfacing with Pre-WS and WS Grid Resource Management Services. Future Generation Computing System, Vol. 23, 2007, No. 2, pp. 252–261.

[3] Padgett, J.—Djemame, K.—Dew, P.: Grid-Based SLA Management. In Proc. of the European Grid Conference (EGC), Amsterdam, The Netherlands, 2005.

[4] Stantchev, V.—Schröpfer, C.: 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.

[5] Andrieux, A. et al.: Web Services Agreement Specification (WS-Agreement). Tech. report, 2007. Availaible on: `https://forge.gridforum.org/projects/graap-wg/`.

[6] AssessGrid. Availaible on: `http://www.assessgrid.eu`.

[7] EU-Brein. Availaible on: `http://www.eu-brein.com/`.

[8] SLA@SOI. Availaible on: `http://sla-at-soi.eu/`.

[9] Vázquez, C.—Huedo, E.—Montero, R. S.—Llorente, I. M.: Federation of Teragrid, EGEE and OSG Infrastructures Through a Metascheduler. Future Generation Computer Systems, Vol. 26, 2010, No. 7, pp. 979–985.

[10] Roy, A.—Sander, V.: Grid Resource Management. Chapter GARA: A Uniform Quality of Service Architecture. pp. 377–394. Kluwer Academic Publishers, 2003.

[11] Siddiqui, M.—Villazón, A.—Fahringer, T.: Grid Capacity Planning with Negotiation-Based Advance Reservation for Optimized QoS. In Proc. of the 2006 Conference on Supercomputing (SC), Tampa, USA, 2006.

[12] Waldrich, O.—Wieder, Ph.—Ziegler, W.: A Meta-Scheduling Service for Co-Allocating Arbitrary Types of Resources. In Proc. of the 6th Intl. Conference on Parallel Processing and Applied Mathematics (PPAM), Poznan, Poland, 2005.

[13] Dobber, M.—van der Mei, R.—Koole, G.: A Prediction Method for Job Runtimes on Shared Processors: Survey, Statistical Analysis and New Avenues. Performance Evaluation, Vol. 64, 2007, No. 7-8, pp. 755–781.

[14] Dinda, P. A.: The Statistical Properties of Host Load. Scientific Programming, Vol. 7, 1999, No. 3-4, pp. 211–229.

[15] Jin, H.—Shi, X.—Qiang, W.—Zou, D.: An Adaptive Meta-Scheduler for Data-Intensive Applications. Intl. Journal of Grid and Utility Computing, Vol. 1, 2005, No. 1, pp. 32–37.

[16] Tomás, L.—Caminero, A.—Carrión, C.—Caminero, B.: Network-Aware Meta-Scheduling in Advance with Autonomous Self-Tuning System. Future Generation Computer Systems, Vol. 27, 2011, No. 5, pp. 486–497.

[17] Sulistio, A.—Cibej, U.—Prasad, S. K.—Buyya, R.: GarQ: An Efficient Scheduling Data Structure for Advance Reservations of Grid Resources. Intl. Journal of Parallel Emergent and Distributed Systems, Vol. 24, 2009, No. 1, pp. 1–19.

[18] Castillo, C.—Rouskas, G. N.—Harfoush, K.: On the Design of Online Scheduling Algorithms for Advance Reservations and QoS in Grids. In Proc. of the Intl. Parallel and Distributed Processing Symposium (IPDPS), Los Alamitos, USA, 2007.

[19] Theilmann, W.—Baresi, L.: Multi-Level SLAs for Harmonized Management in the Future Internet. Chapter Towards the Future Internet. pp. 193–202, IOS Press, 2009.

[20] Brandic, I. et al.: Advanced QoS Methods for Grid Workflows Based on Meta-Negotiations and SLA-Mappings. In Proc. of the 3rd Workshop on Work ows in Support of Large-Scale Science, Austin, USA, 2008.

[21] Ejarque, J. et al.: Exploiting Semantics and Virtualization for SLA-Driven Resource Allocation in Service Providers. Concurrency and Computation: Practice and Experience, Vol. 22, 2010, No. 5, pp. 541–572.

[22] Waeldrich, O. et al.: WS-Agreement Negotiation Ver. 1.0. Tech. report, 2011.

[23] Davide Lamanna, D.—Skene, J.—Emmerich, W.: Slang: A Language for Defining Service Level Agreements. In Proc. of the Intl. Workshop of Future Trends of Distributed Computing Systems, Los Alamitos, USA, 2003.

[24] WSLA: Web Service Level Agreements. Availaible on: `http://www.research.ibm.com/wsla/`.

[25] Parkin, M.—Badia, R. M.—Martrat, J.: A Comparison of SLA Use in Six of the European Commissions FP6 Projects. Tech. report TR-0129, 2008.

[26] WSAG4J – WS-Agreement Framework for Java. Availaible on: `http://packcs-e0.scai.fraunhofer.de/wsag4j/`.

[27] NextGrid – Architecture for Next Generation Grid projects. Availaible on: `http://www.nextgrid.org/`.

[28] Caminero, A.—Rana, O.—Caminero, B.—Carrión, C.: Network-Aware Heuristics for Inter-Domain Meta-Scheduling in Grids. Journal of Computer and System Sciences, Vol. 77, 2011, No. 2, pp. 262–281.

[29] di Stefano, A.—Morana, G.—Zito, D.: A P2P Strategy for QoS Discovery and SLA Negotiation in Grid Environment. Future Generation Computer Systems, Vol. 25, 2009, No. 8, pp. 862–875.

[30] Tomás, L.—Caminero, A.—Carrión, C.—Caminero, B.: 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.

[31] Armstrong, D.—Djemame, K.: Towards Quality of Service in the Cloud. In Proc. of the 25th UK Performance Engineering Workshop, Leeds, UK, 2009.

[32] Gridsphere. Availaible on: `http://www.gridsphere.org/`.

Javier Conejero is a Ph. D. student in computer science at the University of Castilla-La Mancha, Spain. He received his B. E. and M. E. in computer science from the University of Castilla-La Mancha. He worked at CERN for one year in WLCG software development and management. He has been working with Service Level Agreements within Grid environments since 2010. His current research interests include QoS with SLAs in Grid and Cloud environments.

Luis Tomás is a Ph. D. student in computer ccience at the University of Castilla-La Mancha, Spain. He received his B. E. and M. E. in Computer Science from The University of Castilla-La Mancha (Albacete, Spain) in 2007 and 2009. He has been working with resource management and job scheduling for Grid environments since 2007. His current research effort is on efficient meta-scheduling in advance in Grids to provide QoS to users. His work considers QoS in terms of completion time guarantees.

Blanca Caminero is an Associate Professor of computer architecture and technology at the Computing Systems Department (The University of Castilla-La Mancha). She holds a Ph. D. Degree in Computer Science from The University of Castilla-La Mancha, and her current research interests are in QoS support and meta-scheduling in Grids and Clouds. She is a member of the IEEE.

Carmen Carrión is an Associate Professor of computer architecture and technology at the Computing Systems Department at The University of Castilla-La Mancha. She holds a Ph. D. Degree in Physics from The University of Cantabria, and her interests include meta-scheduling and QoS in Grids and Clouds.