Computer Science • 13 (4) 2012 http://dx.doi.org/10.7494/csci.2012.13.4.183

Axel Tenschert
Roland Kübert

SLA-BASED JOB SUBMISSION AND SCHEDULING WITH THE GLOBUS TOOLKIT 4

Abstract *High performance computing is nowadays mostly performed in a best effort fashion. This is surprising as the closely related topic of grid computing, which deals with the federation of resources from multiple domains in order to support large jobs, and cloud computing, which promises seemingly infinite amounts of compute and storage, both offer quality of service (QoS), albeit in different ways. Long-term service level agreements (SLAs), which require the establishment of SLAs long in advance of their actual usage, seem a promising way for the offering of QoS guarantees in an HPC environment in a way that is not disruptive to the business models employed today. This work uses the long-term SLA approach as a basis for the provisioning of service levels for HPC resources and presents an SLA management framework to support this. Flexibility is provided by providing SLAs with different service levels, support for which is integrated into job submission and scheduling. The SLA management framework can, on a high level, be used in a generic fashion and an implementation is presented that is evaluated against a motivating scenario.*

Keywords service level agreements, service-oriented architecture, grid computing, semantic matching

1. Introduction

High performance computing (HPC) has traditionally been provided in a best effort manner, meaning that no quality of service (QoS) guarantees were given to users. Developments coming out of the grid computing domain and, more recently, efforts from the cloud computing domain, show the possibility to provide QoS guarantees for HPC-like scenarios. While cloud computing, with its notion of basically infinite resources, comes in some aspects close to state of the art HPC resources, it simply cannot be used to solve certain problems. This is mainly due to the fact that specialized network interconnects, which are a core feature of HPC resources, are as of yet not possible in cloud scenarios.

For the description, provisioning and enforcement of QoS terms, service level agreements (SLAs) have been a popular tool. Thus, they lend themselves for usage in the HPC domain as well. Especially in grid computing, SLAs enjoy widespread usage and countless approaches have been proposed and several implemented. For cloud computing, SLAs are not that prominent and exist in a much simpler form than those proposed for grid computing applications. In the HPC area, little has been done regarding SLAs as there was no real requirement since best effort service prevailed.

In order to realize QoS in an HPC environment, SLAs can only serve as a tool. HPC providers mainly need to answer the questions of which service levels they want to provide and how they can be implemented. The first question needs to be addressed by the HPC provider in conjunction with its users. Kübert [20, 19] and Kübert and Wesner [21, 22] propose the usage of SLAs which are not dynamically negotiated on a per-job basis but which are prepared well in advance before job submission. These "long-term SLAs" are a feasible approach to provide service levels in a manner that is not disruptive to contracting and user management that is performed at HPC providers today. Apart from relying on long-term contracts, a key point of this approach is that HPC providers will most likely procure a small number of service levels; this runs contrary to work using per-job negotiation as well, which normally results in a potentially huge number of different service levels. This approach, however, resides rather on the conceptual level and does not prescribe any details of actual implementation.

Implementing service level support in an HPC environment requires solving two problems: implementation on the middleware layer and above and implementation on the resource level. While the former can and should be addressed in a generic fashion, the latter can only be addressed depending on how the first question mentioned above, the how and which of service levels, has been answered. Different service levels require different implementations on the resource layers and can hardly be generalized; Kübert and Wesner have, for example, investigated the influence of cancelling best effort based jobs in order to ensure wait time guarantees [23].

This work proposes a solution for the open question previously mentioned, namely how to realize a generic framework for the support of service levels in an HPC environment. This is done by analysing a specific scenario that is motivated from a use

case occurring in HPC. For the specific scenario, the resource layer implementation is presented as well. While the former part can be used without changes for other scenarios, the latter part needs to be adapted to the case at hand.

Outline The remainder of this article is organized as follows. Section 2 gives account of related work. Section 3 discusses requirements by both service providers and users through an example scenario, motivated from the real world. Section 4 presents the high-level architecture of the proposed solution while section 5 discusses implementation details. Section 6 evaluates the presented concept against the requirements derived from the scenario. Section 7 concludes the work and gives an outlook on future work.

2. Related work

Service level agreements have been used for various purposes in the grid and distributed computing area. Most of this work is not considering HPC in its true sense using dedicated supercomputers. While the aspect of job scheduling itself is quite often treated in isolation (see for example [43, 44, 17, 16, 29]), there is no investigation of overarching solutions encompassing SLA management and job submission and scheduling at the same time. If SLA management is addressed, this is mostly done on a high level and does not extend to the middleware or resource layer. During job submission and scheduling SLA information is assumed to just "be there".

Advance reservation is in general a concept that is suitable for providing certain QoS guarantees and implementations exist that can be readily used, for example the Maui Scheduler[1]. However, advance reservation is best applicable to scenarios which require precise execution times, for example for co-allocation of resources [25] [24]. Co-allocation is, however, only one possible service level (see for example Wesner [42], who proposes additional service levels). Furthermore, advance reservation does not scale well, making its use difficult for current HPC resources, even more so for future exascale systems [9][2].

Service level agreement management has been a popular research area in general. It has been broadly covered by various European research projects. The ICT-funded research projects NextGRID [26], Business Experiments in GRID (BEinGRID) [6], Business objective driven reliable and intelligent grids for real business (BREIN) [8] and Interactive Realtime Multimedia Applications on Service Oriented Infrastructures (IRMOS) [2]. To exemplify the approaches taken we examine the approaches taken by BEinGRID and BREIN.

Figure 1 shows a high-level view of the different phases during the lifetime of an SLA that have been identified in BEinGRID. Different requirements were identified

[1] `http://www.adaptivecomputing.com/resources/docs/maui/`

[2] The Cray XE6 "Hermit" in use at the High Performance Computing Center in Stuttgart since December 2011 and delivering a peak performance of "only" 1.045 petaFLOPs has already 3,552 compute nodes with 113,664 cores [14].

from this: SLA template specification, Publication & Discovery, Negotiation, Provider Resource Optimisation, Monitoring, Re-negotiation, Evaluation and Accounting. BEinGRID provides no ready-to-use SLA management framework addressing these requirements but rather composable components addressing specific use cases for SLA negotiation, monitoring and evaluation, violation notification and accounting.

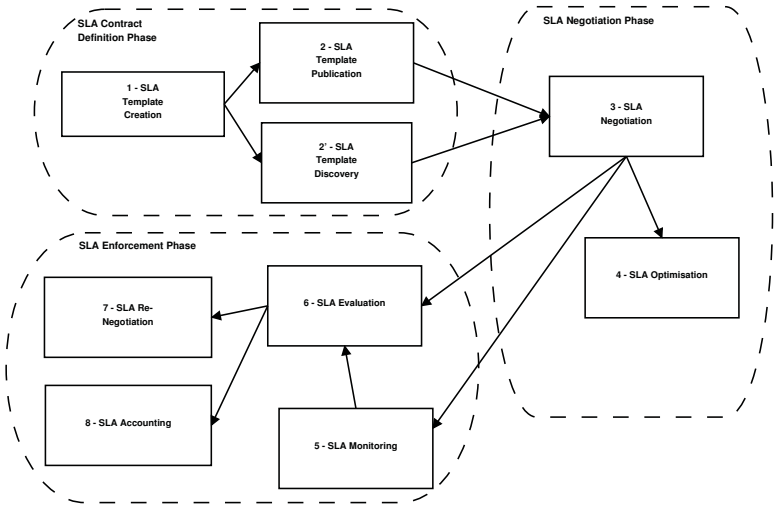

Figure 1. SLA management as seen by the BEinGRID project.

From figure 1 it can be seen that contract definition and negotiation are very important in BEinGRID's approach. This is also obvious in a video presentation of BEinGRID's SLA architecture[3]. Dynamic negotiation of SLAs per individual compute job, however, is an approach that is very unlikely to appeal to users in an HPC environment. BREIN's approach is similar in regard to dynamic negotiation, however the SLA management framework proposed by BREIN is much more complex (see Figure 2).

The SLAs used in BEinGRID were based on the WS-Agreement specification [36] defined by the Open Grid Forum [37]. BREIN used SLAs that were composed mainly according to WS-Agreement but parts were taken from WSLA[15], mostly related to monitoring. While WS-Agreement basically only defines "a language and a protocol for advertising the capabilities of service providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime", an abstract architecture is specified as well. Battré, Kao and Voss have presented an implementation of WS-Agreement in the grid middleware Globus Toolkit 4 (GT4) [5]. This implementation allows for SLA negotiations per job and validates SLAs proposed

[3] `http://www.youtube.com/watch?v=moKSso2gN8w`

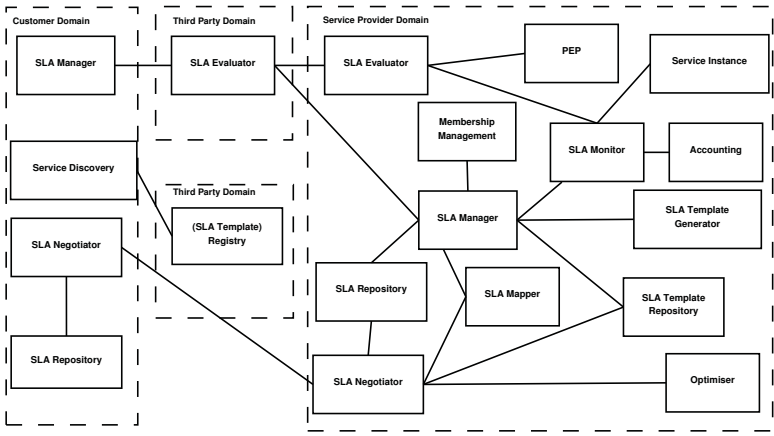

Figure 2. SLA management as seen by the BREIN project.

by customers against constraints specified in a template. Furthermore, it is checked that SLAs can fit into a schedule by the planning based resource manager OpenCCS[4].

Similar approaches have been taken by research projects related to the German grid initiative D-Grid that have investigated SLA management, for example by the projects FinGrid [10] and SLA4D-Grid [30].

Regarding SLA management specifically for HPC environments, an architecture has been proposed by Koller [18]. The architectural approach is complex and in total proposes more than thirty different components. As with the approaches discussed previously, the creation and negotiation of agreements is an important part in this architecture as well. No implementation is discussed in this work.

Most of the projects and approaches mentioned above make use of the Globus Toolkit grid middleware for implementation of their components [35]. Furthermore, they use services already provided by the middleware, for example a directory service, job submission service, notification mechanisms etc. Two other popular middlewares are UNICORE [38] and gLite [32]. Globus Toolkit's current version is 5.x; however, as this version has done away with the Java-based web services supported previously, many sites still run version 4.x in production use, for example the High Performance Computing Center Stuttgart (HLRS)[13], which uses this version even for access to its new Hermit supercomputer inaugurated in December 2011 [14], or the German national grid initiative D-Grid [31] and several related research projects. Hence, this work focuses on GT 4 in order to support a job scheduling in a state of the art HPC environment.

If so many approaches to SLA management, partly even implementations, already exist, a question poses itself: why "yet another SLA management framework"?

[4] `https://www.openccs.eu/core/`

There are a number of answers to this question. Firstly, looking at HPC providers today, there is basically no support for service levels yet, even though service levels have been widely used in other fields which are somewhat closely related to HPC. A reason for this can be found in the overly complex SLA management frameworks (SLAMFs) that have been proposed, for example in the projects mentioned above. The reason for this complexity lies in the fact that these solutions are generic in nature. Therefore, it is difficult to reduce an SLAMF to a core of specialized components. Overly high complexity is, however, even found in specialized SLAMFs, for example in [18]. Implementation, setup and (interdependent) configuration of such a number of different components is a gargantuan task; the adaptation of the scheduler then still remains! It is therefore understandable that the uptake of these solutions is hardly happening at all.

The complexity even of targeted solutions can be explained through the complex SLA lifecycle proposed by the TeleManagement Forum (TMF) [3]. Most SLAMFs are based on this lifecycle, which divides an SLA's life time into six phases: Development, Creation, Provisioning, Execution, Accounting, Assessment and Termination. Even though there are other, slightly simpler lifecycles, for example [28], the TMF's lifecycle seems to be a de-facto standard. SLAMFs generally cover the whole lifecycle and are, therefore, highly complex. We assume that the development, creation, provisioning, accounting and assessment tasks are better treated on their own, potentially off-line, and thus we focus on the execution task. This allows for the development of a much more targeted SLA management framework.

3. Scenario and requirements

As a motivating scenario we present a semantic matching use case. Semantic matching is the process which identifies nodes in multiple graph structures which semantically correspond to one another. The amount of data processed when performing semantic matching varies with the concrete application but this is usually not an issue. However, the reasoning process itself is computationally expensive and time consuming. For instance, in August 2011 within the scope of the large triple store challenge [40] the loading and querying of about one trillion RDF triples [39] was processed. The execution took nearly 338 hours for an average rate of 829.556 RDF triples per second running on 80 cores. Hence, such processes will be distributed whenever it is possible, as data set size is expected to grow. The semantic matching used in this scenario is based on the forward-chaining based matching approach described by Weaver and Hendler [41]. Mostly, the reasoning is not a time critical process and a user is satisfied to run the related computational jobs in best effort manner. However, at times computations might be required in a certain time frame. During the course of work there are various situations which require the preferential use of HPC resources: data needed for presentations or publications, for example, or input that is urgently required for project deliverables which are often updated very close to the documents' deadlines. Reasoning on semantic data is usually performed in different steps which

build on each other. Each process requires data from the previous process as input. A usual process sequence may consist of several steps such as lexical matching, taxonomy matching and non-taxonomy matching.

We assume an expert is able to perform a complete matching cycle step by step in one working day if the individual matching steps are executed in a timely manner. This is valid for data sets up to a certain size only, of course, but without loss of generality we focus on data sets which require not more than a full day for complete processing. Firstly, the lexical matching step is initialized by the expert. The expert needs to wait for the results of this step and afterwards the next matching step can be started if the expert decides to go on with the matching. The domain expert's task in between the steps is the analysis of the output of each step and the decision if the following step is even to be performed. Through the distribution of the matching tasks on available computing resources complete matching and the afterwards performed reasoning can easily be performed within a fixed time-frame if the amount of waiting time for each job execution is known as well. The user therefore requires to be able to perform his computations not only in a best effort manner but also with guaranteed waiting times.

Table 1 shows the requirements from the user's side for this scenario. In order for an HPC provider to address these requirements, we use the long-term SLA approach proposed by Kübert [20, 19] and Kübert and Wesner [21, 22]. In a nutshell, this approach means that an HPC provider procures a small number of service levels only, maybe even just one service level in addition to the usual best effort service level. Users contract each service level they want to use with the provider. This is done well in advance of actual job submission, just like it is done at HPC providers today. However, where there normally is only one so-called "user agreement" in place between HPC provider and customer, the different service levels, which are described in service level agreements (SLAs) can be thought of as multiple user agreements where each user agreement is equivalent to a service level. As with the user agreements currently in place, this approach assumes that these SLAs are contracted on a long-term basis, for example for various years, thus "long-term SLAs". SLA negotiation, which is usually one of the most complex topics of SLA management, is thus treated as a precursor to the actual scenario, greatly simplifying the resulting management framework.

Service levels that are relevant for HPC users have to be elicited by a provider from their users as different user groups will require different service levels. For our case of a semantic reasoning use case and the user requirements listed in Table 1, we assume that the HPC provider offers, besides best effort based access, a service level that gives a maximum waiting time guarantee for jobs submitted in relation to this service level. For the sake of simplicity, we will call the best effort service level from here on "bronze level" and the service level with guaranteed waiting time "silver level".

Table 2 shows the requirements from the provider for the scenario stated above. These are mainly focused on the provider being able to provision different service levels.

Table 1
Scenario user requirements.

Req. No.	Definition	Scenario Example
RC1	Management of large data sets	Semantic matching of several large data sets by consideration of extreme large data volumes such as the use of more than one trillion RDF triples
RC2	Execution of a reasoning task	Execution of the forward-chaining based reasoning approach dealing with high amounts of data
RC3	Latency handling	Dealing with interdependent inference procedures by keeping latencies low
RC4	Time efficiency of reasoning	A restricted time frame
RC5	Allocation of compute resources	A customer requiring HPC resources
RC6	Configuration of compute resources	Solving a specific reasoning strategy
RC7	Contracting of different SLAs	User is able to contract for multiple service levels
RC8	Obtaining accounting information regarding SLAs	User is able to receive detailed information for the usage of the contracted service levels

Table 2
Scenario provider requirements.

Req. No.	Definition	Scenario Example
RP1	Procurement of different service levels	The provider is able to offer for example a best effort based service and one with extended guarantees
RP2	Validation of jobs submitted against their alleged service level	Users might submit jobs to a service level they are not allowed to use
RP3	Job scheduling in accordance with prepared service levels	Job scheduling is performed as envisioned by the provider in accordance with the service levels
RP4	Accounting and billing respecting service levels	Jobs need to be accounted against their service level, taking potential bonuses and rewards into account

For this scenario, we assume that the service levels are, in detail, as follows:

- Best effort (bronze)
  - No limit on job size.
  - Maximum execution time of 24 hours.
  - No limit on jobs in the queue at the same time.
  - No limit on how many jobs are executed simultaneously.
  - No guarantees on waiting time.
- Guaranteed waiting time (silver)
  - Maximum job size 200 cores[5].
  - Maximum execution time of 4 hours.
  - At most 2 jobs in the queue at the same time per user.
  - At most 1 job per use is executed simultaneously.
  - Waiting time is guaranteed to be less than 1 hour.

As presented the bronze level differs from the silver level by the provided resources such the maximum job size. In case the provided resources do not fit the scenario requirements, the SLA level needs to be changed. Further details, for example regarding the price of computational units, which is usually given in core hours, are of no interest for the purposes of this work. One can, of course, assume that the price per computational unit is significantly higher for the silver service level.

4. Architecture

This sections proposes a generic architecture that enables the use of SLAs for grids, clusters and HPC resources. The main influences for the architecture are the requirements presented in section 3. Additionally, the architecture is already aligned with how resource access is usually realized today. Thus, the architecture is geared towards the specifics of the Globus Toolkit 4 (GT4), which is a grid middleware that is employed very often for providing high-level access to grids, clusters and HPC resources. Even though GT4 has been succeeded by the follow-up version 5, it is still in widespread production use.

GT4 is a modular middleware and the part that enables users, among other things, to submit, monitor and cancel remote jobs is called Grid Resource and Allocation Management (GRAM). It is not a job scheduler in itself but provides a single point of access for communication with different job schedulers and resource managers [1]. Specifically, the focus of the presented solution lies on the web service realization WS-GRAM, which is shown in Figure 3. There are two web services that make up the heart of WS-GRAM, *ManagedJobFactory* and *ManagedJob*. The *ManagedJobFactory* service exposes an interface for the creation of jobs for a local scheduler. Submitted jobs are exposed as resources of the *ManagedJob* service. This service can be queried in order to monitor the status of a job, terminate a job etc.

[5]This means 10% of the used cluster.

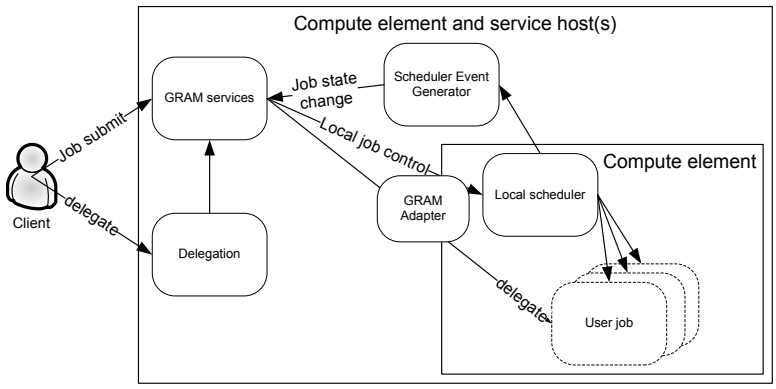

Figure 3. WS-GRAM components [33].

A simple job submission scenario using WS-GRAM follows the sequence shown in Figure 4: the client invokes the *ManagedJobFactory*'s *createManagedJob* operation, which results in a *ManagedJob* resource being created and it's endpoint reference (EPR) being returned to the client. GRAM schedules the job with the local scheduler and notifies the client that the job is pending. On subsequent events — scheduling of the job, job start, job end etc. — GRAM notifies the client. Explicit termination of resources is not necessary.

The architectural requirements for realizing SLA-based job submission are quite straightforward if we adhere to a service-oriented architecture (SOA) approach. Section 3 stated a number of requirements which we can translate quite easily to services and operations.

Requirements RC1, RC4 and RC5 are generic enough that we can presume that they are fulfilled by GT4 already. Requirement RC2 and RC3 require support for timing parameters and requirement RC6 the provisioning of a specific compute environment tailored to the user. RC6 can be seen as a parameter for job submission that exists independently of the service levels silver and bronze and which allows the user to specify certain requirements on software (e.g. version of semantic reasoning library) or hardware (specific CPU and RAM requirements). RC2 and RC3 are requirements on timing and thus result in different QoS classes (silver and bronze in the example scenario). RC7 and RP1 mean that the system needs to be able to differentiate between service levels in a generic way. RC8 has a similar meaning but requires an accounting and billing system that is capable of differentiating between service levels as well; RP4 is addressed through this on the provider side. While from the remaining requirements RP2 can be addressed on the middleware layer, RP3 needs to be realized on the resource layer and is basically the most complex part to realize. It is, however, not a part of the generic SLA management framework but part of the underlying implementation which is always specific to the actual service levels provided.

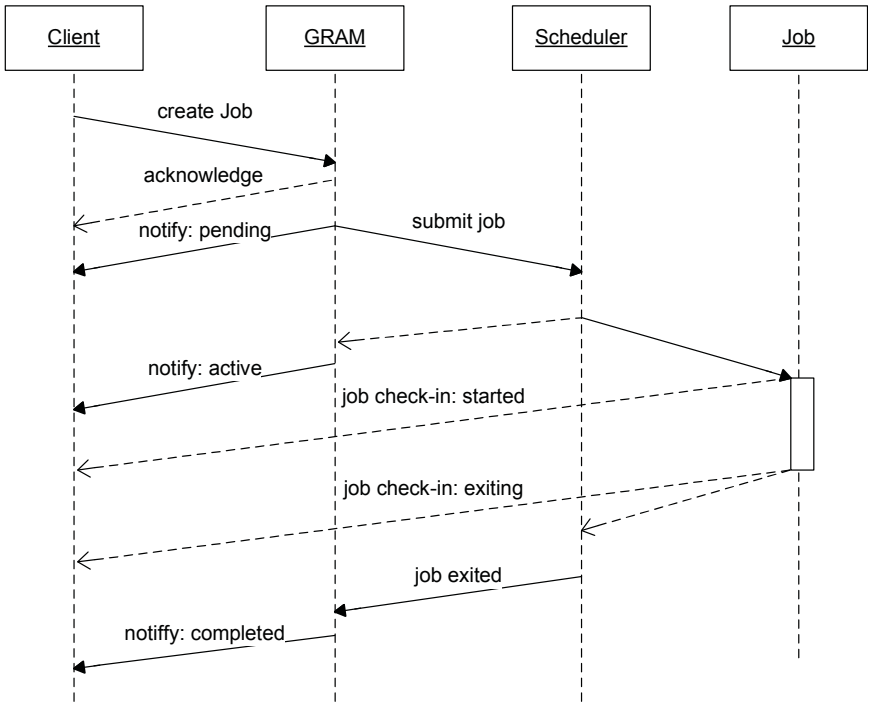

Figure 4. WS-GRAM protocol sequence [33].

A high-level architecture that supports the usage of SLAs is shown in Figure 5. The main component is the SLA Manager, which is responsible for the management of SLA-related data; it uses a *R*epository where the complete SLAs and relevant information are stored. Furthermore, the SLA Manager acts as a policy decision point (PDP): when a job is submitted by a client to the *M*anagedJobFactory, the factory relays SLA-related data to the SLA Manager. The SLA Manager analyzes the data and takes a decision that is communicated back to the factory. Multiple PDPs can be activated at the same and their decision can result in various actions to be enforced (specifically for GT4, the algorithm "AllowOverride", for example, results in one allow received from multiple PDPs to override one or more denies, while "DenyOverride" works the other way round).

If the factory as a policy enforcement point (PEP) finally accepts a job, it is submitted to the scheduling system. Depending on the actual realization of the scheduling, the *S*cheduler might communicate with the SLA Manager in order to obtain relevant data from an SLA associated with a specific job. The SLA Manager is the central point regarding communication of SLA related information to the user. Accounting information, for example, can thus be queried through the SLA manager which can obtain information from other internal components on the provider side.

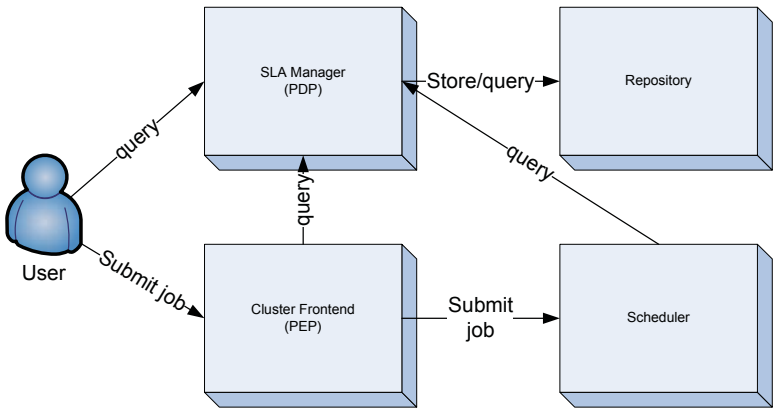

Figure 5. High-level SLA management components.

5. Implementation

The previous section described the high-level architecture used to submit and schedule jobs in relation to SLAs for a grid middleware. This section elaborates on implementation details and describes a proof of concept realization of the architecture.

5.1. Submitting and receiving jobs in relation to SLAs

For the submission of a job related to an SLA, a client needs to be able to have an overview of SLAs which are in place for an HPC provider. The client can either implement its own SLA management stack or can use services provided by the provider. For the sake of simplicity, we assume that SLA-related information is not stored on the client side but is queried on demand from the provider's *SLA Manager* component. The client is provided with a GUI tool that can be used to submit jobs in reference to contracted SLAs as shown in Figure 6. For each of the providers listed in the first text box, SLAs that can be used will be listed in the text box below. The client can then select a prepared job description file that shall be submitted for the given SLA and can finally submit the job.

In the screenshot shown the client is connected to three different HPC providers. It is sufficient for the client to, for example, configure the application with the URLs of the SLA Manager component of each provider that shall be used. SLA information can then be queried dynamically from each provider's SLA Manager. Figure 6 shows three SLAs in place, namely "Gold", "Silver" and "Bronze", which the client application has queried from the provider's *SLA Manager* service. The user can select any one of these when submitting a job.

Practically, the addition of SLA-related information to the job submission is performed by adding a reference — in our case, the unique id of the SLA — to the *MessageContext* when communicating with the *ManagedJobFactoryService*, as shown

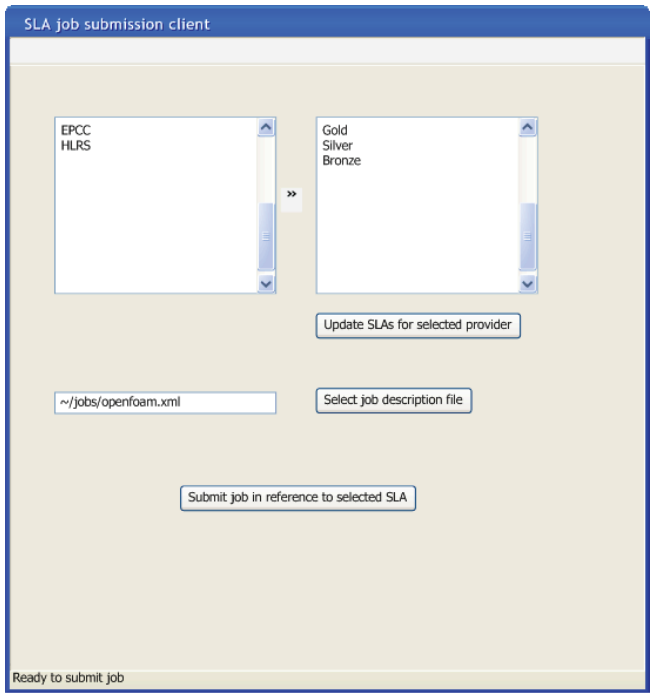

Figure 6. SLA selection and job submission using a GUI client.

in Listing 1. This is certainly not the only way to communicate the SLA id between customer and service provider, but it is a simple and straightforward solution. Both parties of course need to use the same way of conferring and expecting the SLA id in a job submission message.

```
1  private EndpointReferenceType getFactoryEpr(String contact,
2        String factoryType) throws Exception {
3    URL factoryUrl = ManagedJobFactoryClientHelper.getServiceURL(contact)
4      .getURL();
5
6    EndpointReferenceType epr = ManagedJobFactoryClientHelper.
         getFactoryEndpoint(factoryUrl,
7      factoryType);
8
9    // SLAID Start
10   ReferenceParametersType parameters = epr.getParameters();
11   MessageElement slaId = new MessageElement("http://www.hlrs.de/namespaces/
         services/SLA", "slaId");
12   slaId.setValue("6C4E20EA7F000101B05FCB518079B0");
13   parameters.add(slaId);
14   // SLAID End
15
16   return epr;
17 }
```

Listing 1: Adding an SLA id to the factory's message context.

ReferenceParameters are an optional element of an Endpoint Reference and can easily be extracted from the SOAP header on the recipient side [7].

The specification of the job description to be submitted to the provider is taken from a file which the user can select and whose contents can be viewed/edited in the submission client as well. Globus uses Resource Specification Language (RSL), a self-developed XML schema, to allow users to specify complex jobs [11]. An example that runs the executable */bin/sleep* with the argument *60* and specifies custom files for standard output and standard error is given in Listing 2.

```
1 <job>
2     <executable>/bin/sleep</executable>
3     <argument>60</argument>
4     <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
5     <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
6 </job>
```

Listing 2: A simple sleep job in RSL.

If the user has selected a service level and a job description file, the corresponding job will be submitted to the provider once the user activates the *Submit* button shown in the bottom right corner in Figure 6.

5.2. Receiving SLA information on the provider side

The SLA id contained in the SOAP header of the job submission message is extracted on the provider side; in the implementation presented here, the following checks are implemented:

- The certificate identifying the user is checked against one stored with the SLA in order to confirm if the user is allowed to use this certificate; without this check, any user who would somehow obtain an SLA id could use it to compute on the cost of other customers.
- The SLA is checked for timely validity, that is if the run-time of the SLA has expired or not.
- A customer is not allowed to have more than one job queued in the prioritized service level; there is no limit on best effort jobs.
- All prioritized SLAs only allow a maximum wall time of 4 hours; if the *maxWall-Time* property in the job description exceeds this time, the job is not allowed to run.

Both the selection of the parameters that are checked and the values of these parameters are arbitrary and only serve to illustrate the possibility of how easily checks of a job submission's property can be validated against an SLA.

5.3. Securing the ManagedJobFactory service

Calls to the ManagedJobFactory service need to pass the SLA PDP, additional to any other PDPs that are in place as well. GT4 uses so-called "security descriptors" for configuring different security properties, for example credentials, authentication and

authorization mechanisms etc. [34] Therefore, the SLA PDP has to be added to the security descriptor for the *M*anagedJobFactory [6] as shown in Listing 3.

PDPs are configured in an authorization chain and are evaluated in turn to finally arrive at a permit or deny decision. If the PDPs are combined with deny override, as is shown in Listing 3, all PDP have to arrive at a permit decision in order for the complete chain to authorize a request. The SLA PDP is, as first PDP, evaluating if the request specified a correct SLA id which is valid for the user presenting it, checking that the referenced SLA allows for a submission of a job at the current time (see the section 5.2). If the SLA PDP denies the request, no further evaluation is necessary. However, if the SLA PDP permits the request, the Gridmap PDP, a default Globus PDP, is invoked, to see if the user calling the service can be mapped to a local user. Only if this PDP arrives at a permit, the job can finally be queued.

```
<serviceSecurityConfig>
<methodAuthentication>
    <method name="createManagedJob">
        <auth-method>
            <GSISecureConversation/>
            <GSISecureMessage/>
            <GSISecureTransport/>
        </auth-method>
    </method>
</methodAuthentication>
<authzChain combiningAlg="DenyOverride">
    <pdps>
      <interceptor
       name="slapdp:de.hlrs.gt4.pdp.sla.Gt4SlaPdp" />
      <interceptor
       name="gridmap"/>
    </pdps>
 </authzChain>
<reject-limited-proxy value="true"/>
</serviceSecurityConfig>
```

Listing 3: Security descriptor for the *ManagedJobFactory*.

The PDP extends the interface *org.globus.wsrf.security.authorization.PDP* and has one important operation, *isPermitted*. Listing 4 shows how the SLA PDP is implemented in a high-level pseudo code. The caller is passed to this operation by GT4, as well as the message context and the name of the operation called. The SLA id, which has been embedded by the user in the message context, is extracted and the SLA id and caller id are passed on to the SLA Manager in order for it to assess if the SLA id is valid and if a job can be submitted at the current time for the specified contract. The PDP then relays the SLA Manager's decision back to the caller.

```
  public boolean isPermitted(Subject peerSubject, MessageContext context,
      QName operation) throws AuthorizationException {

    boolean isPermitted = true;

```

[6] The default location is
`$GLOBUS_LOCATION/etc/globus_wsrf_gram/managed-job-factory-security-config.xml`

```
6       String caller = SecurityManager.getManager(
7           (SOAPMessageContext) context).getCaller();
8
9       String operationName = operation.getLocalPart());
10
11      // Only execute authorization check for the createManagedJob method
12      if (operationName.equals("createManagedJob")) {
13        String slaId = getSlaIdFromMessageContext();
14        isPermitted = jobSubmissionPermitted(slaId, caller);
15      }
16
17      return isPermitted;
18    }
```

Listing 4: Realization of the SLA PDP.

5.4. Scheduling jobs in reference to SLAs

The connection between the Globus GRAM web services (*ManagedJobFactory*), to which the users submits its job, and the underlying scheduler is realized with a "Scheduler Interface" realized in the Perl programming language [12].

Globus provides some interfaces for the schedulers Condor, LSF and PBS and a basic fork scheduler which is mainly for testing purposes; only the fork and PBS scheduler interfaces are installed by default [7].

Every scheduler interface has to be provided as a Perl module that subclasses the *Globus::GRAM::JobManager* module which provides an implementation of the base behavior of a job manager. The scheduler interface provides the connection to a specific scheduler. The name of the interface needs to match the scheduler type string (in lower case), thus for the PBS scheduler, this results in the name *Globus::GRAM::JobManager::pbs*. The parent module, *Globus::GRAM::JobManager*, provides several methods of which only the *submit* and *cancel* methods need to be implemented. The PBS script, which has been adapted to this purpose, provides the following four methods:

cancel Cancels a job with a given job id by calling PBS' *qdel* command,

myceil Performs rounding when computing the number of nodes to use in a cluster,

poll Polls the status of a job with a given id by calling PBS' *qstat* command,

submit Constructs a job description file out of the parameters given to the script and submits it by calling PBS' *qsub* command.

The scheduler script receives a job description that is stored in the *ManagedJob*. In the implementation at hand, the SLA id is taken as the name of the queue to which the job is to be submitted; to which queue a job is submitted could of course be computed in other ways. The job description is finally submitted via the *qsub* command to PBS. In this implementation, the service levels are *Silver* and *Bronze* and are mapped to corresponding queues where the Silver queue has a higher priority

[7] The modules can be found in *$GLOBUS_LOCATION/lib/perl/Globus/GRAM/JobManager/*

than the Bronze queue. If necessary, jobs for the *Bronze* queue are cancelled and requeued.

For example, a request reaching the scheduler interface where the SLA reference is specified as *Silver* would be submitted to the queue named *Silver* with *qsub -q Silver...*. The mapping of service levels to queues can be implemented at different levels; for the sake of simplicity it has been implemented as a 1:1 mapping of service level to queue in the *ManagedJobFactory*. The scheduler interface could as well only be passed the SLA id and could itself query the *SLA Manager* for the corresponding service level; as the job submission parameters are already checked against the SLA in the higher level middleware layer, it suggests itself to perform the mapping directly at this stage as well.

6. Evaluation

The scenario introduced in section 3 provided the requirements which are evaluated in this section. For both the user and the provider requirements, we analyse how far the individual requirements have been addressed by the architecture presented in section 4 and the implementation addressed in section 5. Table 3 contains the user requirements, Table 4 the provider requirements.

The described scenario was simple but sufficient for demonstration purposes. Thus, the presented SLA based job scheduling is not restricted to the selected scenario but it is usable for scenarios needing HPC resources, e.g. virtual turbine simulation. In general, the scenario can be extended easily by employing a more complex scheduling by the provider, which is reflected in the corresponding SLAs. The presented job submission by use of SLAs offers the possibility to allocate needed computing resources with an individual configuration to deal with waiting time guarantees.

7. Conclusions and future work

This work has shown how high-level SLA management can be integrated into a common grid middleware, the Globus Toolkit, and, subsequently, into the scheduler. This allows the realization of an integrated solution for the offering, usage and accounting of service level agreements in a grid environment. The scope of this work only allows for a prototypical implementation of the, in total, quite complex solution. Therefore, different tasks remain to be tackled in future work. Firstly, some components might be desirable in addition to the basic solution presented here, for example regarding monitoring of SLA usage, accounting, etc. Secondly, this is in many aspects a proof of concept implementation, therefore it is not directly ready to be deployed in a production environment. Preparing, configuring and packaging the realized components so that they can be used straight away, is therefore a desirable task.

In terms of the scheduling of service levels, we have only presented a simple, queue-based approach. This decision was, once more, due to the proof of concept nature of this work. One can, however, only go so far with a queue-based approach;

Table 3
Addressed user requirements.

Req. No.	Solution for the Scenario
RC1 & RC2	The distribution of the matching and the reasoning enables the execution of matching complex semantic structures and afterwards performing of a forward-chaining based reasoning. However, this scenario expects selected data sets as compatible with similar structure, language or content.
RC3	Latency management for the user is possible through usage of the silver service level, which gives guarantees on waiting time.
RC4	The described scheduling approach is based on long-term SLAs. The SLAs define the amount, type and configuration of usable computing resources as well as the period in which such resources are available for the expert. Through the use of the SLAs the expert is enabled to allocate computing resources as needed for executing urgent tasks in the given time frame.
RC5 and RC6	Due to the presented scheduling approach and the use of tailor made SLAs the needed computing resources as well as the configuration of such resources is handled. The reasoning task is executed by use of the computing resources and configurations defined in the long-term SLA.
RC7	The distinction between different service levels – bronze and silver in this scenario – allow the user to select the most appropriate one for a specific computation.
RC8	The user may receive accounting information regarding the amount of used resources in total and per SLA via the provider's SLA manager service. The billing process will make use of this information as well.

many complex operations, for example advance reservation, are not or not easily possible with queue-based approaches but require a schedule-based approach. Realizing more complex scheduling components is a complex yet interesting work in its own right. However, it becomes possible to combine the presented work with results acquired through the European funded plugIT project [27]. Within the plugIT project graphical models are used to present an available infrastructure of an HPC center, e.g. HLRS [13], in order to perform a semantic matching on the graphical models to create an SLA recommendation. The automatically produced SLA recommendations are sent to the customer as SLA offers to be negotiated.

In general, the work described in this paper is an approach for performing an SLA based job submission usable for receiving required computing resources. Thus the use case scenario becomes possible due to the allocation of the needed resources through the SLA based job submission. The SLA negotiation with the aim to allocate needed resources is related to the above mentioned SLA4D-Grid project [30] developing an infrastructure for SLA management and negotiation usable for grid architectures. Hence, SLA4D-Grid was considered as related work and the results produced in this

Table 4
Addressed provider requirements.

Req. No.	Solution for the Scenario
RP1	The provider has the possibility to offer different service levels as required by the users – best effort (bronze) and guaranteed waiting time (silver) in this scenario.
RP2 & RP3	On job submission, users are authenticated and subsequently authorized. This prevents unauthorized use of SLAs that have not been contracted by a user. For allowed use, the SLA parameters can be partially checked on submission (silver job length less than 4 hours, at most 2 silver jobs allowed to be queued simultaneously, etc.) or during scheduling (only 1 silver job execute concurrently per user).
RP4	HPC providers currently account for resource usage by analysing resource manager logs, either with a custom solution or by using existing tools like pbsacct [4]. Basic accounting for service level usage can be done by storing the service levels for submitted jobs. Furthermore, by analysing the job details it can be easily seen if QoS guarantees have been met, for example regarding guaranteed waiting time, which enables the implementation of penalties/discounts and rewards.

work are of interest for the project as well. Generally, the presented work fits well to customers needs requiring computing resources and it is applicable for HPC centers.

References

[1] GRAM — Globus. `http://dev.globus.org/wiki/GRAM`.
[2] IRMOS project home page. `http://irmos-project.eu/`.
[3] TMF — TeleManagement Forum Homepage. `http://www.tmforum.org/`.
[4] Aveleda A.: pbsacct project home page, 2006.
[5] Battré D., Kao O., Voss K.: Implementing WS-Agreement in a Globus Toolkit 4.0 Environment. pp. 409–418. 2008.
[6] BEinGRID Consortium: BEinGRID project home page, 2009. `http://www.it-tude.com/projects/beingrid`.
[7] Box D., Christensen E., Curbera F., Ferguson D., Frey J., Hadley M., Kaler C., Langworthy D., Leymann F., Lovering B., Lucco S., Millet S., Mukhi N., Nottingham M., Orchard D., Shewchuk J., Sindambiwe E., Storey T., Weerawarana S., Winkler S.: Web Services Addressing (WS-Addressing). Technical Report, W3C, August 2004.
[8] BREIN Consortium: Brein project home page, 2008. `http://www.eu-brein.com/`.

[9] Castillo C., Rouskas G. N., Harfoush K.: On the design of online scheduling algorithms for advance reservations and qos in grids. In *IPDPS*, pp. 1–10. IEEE, 2007.

[10] FinGrid Consortium: FinGrid project home page, 2010. `http://www.fingrid.de/`.

[11] Globus Alliance: GT 4.0 WS GRAM: Job Description Schema Doc, 2004.

[12] Globus Alliance: WS-GRAM Scheduler Interface Tutorial (Perl Module), 2005.

[13] High Performance Computing Center Stuttgart: HLRS — High Performance Computing Center Stuttgart, University of Stuttgart, 2012. `http://www.hlrs.de/`.

[14] Höchstleistungsrechenzentrum Stuttgart: Cray XE6 (Hermit), 2012. `http://www.hlrs.de/systems/platforms/cray-xe6-hermit/`.

[15] Keller A., Ludwig H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manage.*, 11(1):57–81, 2003.

[16] Klusáček D.: Dealing with Uncertainties in Grids through the Event-based Scheduling Approach. In *4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2008)*, pp. 91–98, 2008.

[17] Klusáček D., Rudová H., Baraglia R., Pasquali M., Capannini G.: Comparison of Multi-Criteria Scheduling Techniques. In *Integrated Research in Grid Computing*. Springer, 2008.

[18] Koller B.: *Enhanced SLA mangement in the high performance computing domain*. PhD in Engineering Sciences, Dissertation, Universität Stuttgart,, 2011.

[19] Kübert R.: Providing Quality of Service through Service Level Agreements in a High-Performance Computing Environment. In P. Iványi, B. Topping, eds., *PARENG 2011, 2-nd Int. Conf. on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Ajaccio, France, April 2011. Civil-Comp Press. Paper 52.

[20] Kübert R.: Service Level Agreements for Job Control in Grid and HPC Computing. In Preve N., eds., *Computational and Data Grids: Principles, Applications and Design*, pp. ***–***. Information Science Pub, 2011.

[21] Kübert R., Wesner S.: Service Level Agreements For Job Control in High-Performance Computing. In *IMCSIT*, pp. 655–661, 2010.

[22] Kübert R., Wesner S.: Using service level agreements in a high performance computing environment. *Scalable Computing Practice and Experience*, 12(2):164–177, 2011.

[23] Kübert R., Wesner S.: High performance computing as a service with service level agreements. In *Proc. of the 9th IEEE International Conference on Service Computing (SCC)*, 2012. to appear.

[24] Min R.: Scheduling advance reservations with priorities in grid computing systems. 2001.

[25] Netto M. A. S., Bubendorfer K., Buyya R.: Sla-based advance reservations with flexible and adaptive time qos parameters. In *Proc. of the 5th International Conference on ServiceOriented Computing (ICSOS), vol. 4749 of Lecture Notes*

in Computer Science, pp. 119–131. Springer, 2007.

[26] NextGRID Consortium: Nextgrid project home page, 2008. `http://nextgrid.org/`.

[27] plugIT Consortium: plugIT project, 2011. `http://plug-it.org`.

[28] Redbooks I.: *Service Lifecycle Governance With IBM Websphere Service Registry and Repository*. Vervante, 2009.

[29] Sakellariou R., Yarmolenko V.: *Job Scheduling on the Grid: Towards SLA-Based Scheduling*. IOS Press, 2008.

[30] SLA4D-Grid Consortium: SLA4D-Grid project home page, 2012. `http://www.sla4d-grid.de/`.

[31] The D-GRID GmbH: D-GRID homepage, 2012. `http://www.d-grid-gmbh.de/index.php?id=169`.

[32] The EGEE Project: EGEE homepage, 2009. `http://technical.eu-egee.org/index.php?id=149`.

[33] The Globus Alliance: GT 4.0 WS GRAM Approach. `http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Approach.html`.

[34] The Globus Alliance: Security descriptors. `http://www.globus.org/toolkit/docs/4.0/security/authzframe/security_descriptor.html`.

[35] The Globus Alliance: Globus Toolkit homepage, 2012. `http://www.globus.org/toolkit/`.

[36] The GRAAP WG of the OGF: Web Services Agreement Specification, 2007. `http://www.ogf.org/documents/GFD.107.pdf`.

[37] The OGF: The Open Grid Forum homepage, 2012. `http://www.ogf.org/`.

[38] The UNICORE Forum: UNICORE homepage, 2012. `http://www.unicore.eu/`.

[39] The World Wide Web Consortium: The W3C RDF page, 2004. `http://www.w3.org/TR/rdf-concepts/`.

[40] The World Wide Web Consortium: The W3C Large Triple Store page, 2011. `http://www.w3.org/wiki/LargeTripleStores`.

[41] Weaver J., Hendler J. A.: Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *Proc. of the 8th International Semantic Web Conference*, pp. 682–697, 2009.

[42] Wesner S.: *Integrated Management Framework for Dynamic Virtual Organisations*. Dissertation, Universität Stuttgart, Stuttgart, Germany, 2008.

[43] Yarmolenko V., Sakellariou R.: An Evaluation of Heuristics for SLA Based Parallel Job Scheduling. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 8 p., April 2006.

[44] Yarmolenko V., Sakellariou R.: An evaluation of heuristics for sla based parallel job scheduling. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 8 pp., April 2006.

Affiliations

Axel Tenschert
Höchstleistungsrechenzentrum Stuttgart, Nobelstraße 19, 70569 Stuttgart, Germany,
tenschert@hlrs.de

Roland Kübert
Höchstleistungsrechenzentrum Stuttgart, Nobelstraße 19, 70569 Stuttgart, Germany,
kuebert@hlrs.de

Received: 13.04.2012
Revised: 9.07.2012
Accepted: 3.09.2012