European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

A proposal for a generic Grid scheduling architecture

N. Tonellotto
nicola.tonellotto@isti.cnr.it
Information Science and Technologies Institute
Italian National Research Council
56100 Pisa, Italy

R. Yahyapour
ramin.yahyapour@unido.de
Institute for Robotics Research
University of Dortmund
44221 Dortmund, Germany

Ph. Wieder
ph.wieder@fz-juelich.ded
Central Institute for Applied Mathematics
Research Centre Jülich
52425 Jülich, Germany

CoreGRID Technical Report
Number TR-00015
September 13, 2005

Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

A proposal for a generic Grid scheduling architecture

N. Tonellotto
nicola.tonellotto@isti.cnr.it
Information Science and Technologies Institute
Italian National Research Council
56100 Pisa, Italy

R. Yahyapour
ramin.yahyapour@unido.de
Institute for Robotics Research
University of Dortmund
44221 Dortmund, Germany

Ph. Wieder
ph.wieder@fz-juelich.ded
Central Institute for Applied Mathematics
Research Centre Jülich
52425 Jülich, Germany

CoreGRID TR-00015

September 13, 2005

Abstract

In the last years, several Grid scheduling systems have been implemented to solve "ad hoc" grid problems. In this work we try to identify the common characteristics of three common Grid scheduling scenarios, and we propose a single entity that we call Scheduling Instance that can be used as a building block for the scheduling solutions presented. We identify the behaviour that a Scheduling Instance must exhibit in order to be composed with other instances to build the Grid scheduling systems discussed, and their interaction with other Grid functionalities like information services and the like.

Contents

1 Introduction

"The size of Grids, the different application scenarios, and the diversity of the user and owner policies lead to a difficult allocation problem that cannot be manually executed by the user. This task does not only include the search for suitable resources but also the coordination of the actual allocation times in the selected set of resources. Therefore, an efficient and flexible Grid scheduling system is eventually required to manage the user or application requests. Further, the independent resource providers typically want to maintain control of their Grid resources by use of local management systems. This increases the complexity of the allocation problem, as those local management systems usually do not provide all system information due to their architecture or due to policy restrictions. The same is true for the consideration of the individual job requirements and complex scheduling objectives of the different users. Due to the heterogeneity of a typical Grid, the quality of service and the costs will vary for the different resource providers. This will obviously influence the user preference for a specific resource. Consequently, Grid scheduling significantly differs from the conventional job scheduling on parallel computer systems, which has been addressed frequently in the past" [1].

In general, it can be seen that Grid scheduling is eventually required for many applications. It can also be assumed that not a single Grid scheduling strategy will suit all application scenarios. Therefore, independent and individual implementations will be necessary. The following examples (derived from [2]) depict several scheduling infrastructures that can be used to derive some general concepts about a generic grid scheduling architecture.

1.1 Example: Enterprise Grids

Enterprise Grids represent a scenario of commercial interest in which the available IT resources within a company are better exploited and the administrative overhead is lowered by the employment of Grid technologies. The resources are typically not owned by different providers and are therefore not part of different administrative domains. In this

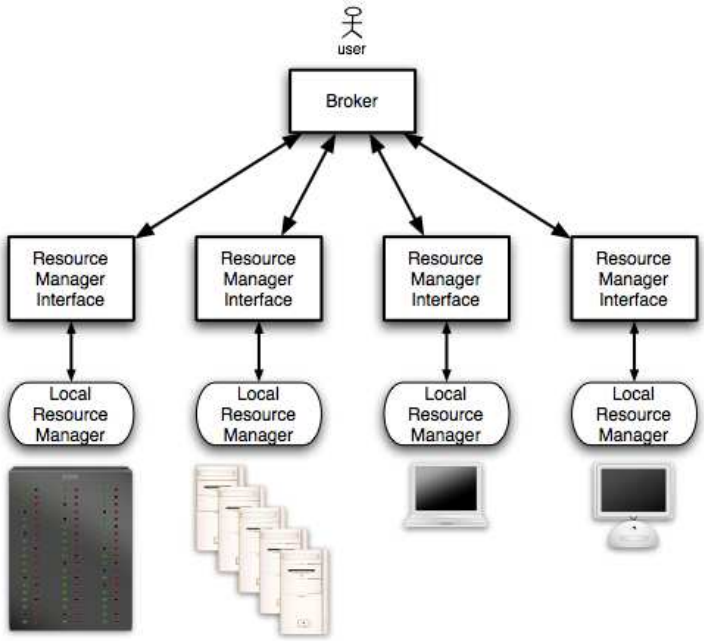

Figure 1: Example of a scheduling infrastructure for Enterprise Grids

scenario we have a centralized scheduling architecture; i.e. a central scheduling instance is the single access point to the whole infrastructure and manages directly the scheduling instances that interact directly with the local resource managers (see Figure 1). Every user must submit jobs to this centralized entity. This architecture can guarantee scheduling decisions in short times, because it does not need to split the problem into sub-problems, to aggregate the decisions, or to forward it to different scheduling instances.

1.2 Example: High Performance Computing Grids

High Performance Computing Grids represent a scenario in which different computing sites, e.g. scientific research labs, collaborate for joint research. Here, compute- and/or data-intensive applications are executed on the participating HPC computing resources that are usually large parallel computers or cluster systems. The total number of participating sites in such a Grid is commonly in the range of tens or hundreds; the available number of processing nodes is in the range of thousands. In this case the resources are part of several administrative domains, with their own policies and rules. In the scenario pictured in Figure 2 we have several institutes that share resources to build an HPC Grid.

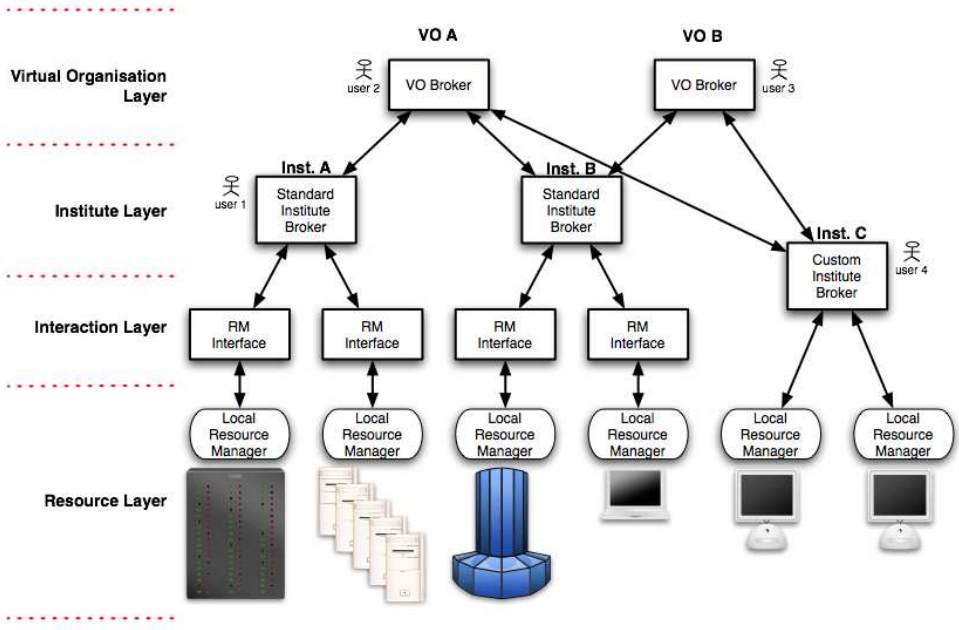

Figure 2: Example of a scheduling infrastructure for HPC Grids

The institutes may organize their own resources in a centralized infrastructure, while the different institutes are connected through Virtual Organizations (VO) in a hierarchical way to build a specific VO scheduling architecture. Every instance on the institute layer can interact with instances in the relevant institute or the responsible instance on the Virtual Organisation layer, depending on the user's affiliation. Please note that no topology is enforced with respect to the possible interactions between the different entities.

A user can submit jobs to the scheduling instances at institute or VO level. The scheduling instances can split a scheduling problem into several sub-problems, or forward the whole problem to different scheduling instances in the same VO.

This infrastructure is fault tolerant, but the hierarchical behaviour can impose a serious performance limit if the number of institutes or VO increases.

1.3 Example: Global Grids

Global Grids might comprise all kinds of resources, from single desktop machines to large-scale HPC machines, which are connected through a global Grid network. This scenario is the most general one, covering both cases illustrated above and introducing a fully decentralized architecture. Every scheduling instance can accept jobs to be scheduled, as Figure 3 depicts.

Like in Peer-to-Peer systems, the fault tolerance is high, but the time needed to find a solution to a scheduling problem can be long and unpredictable

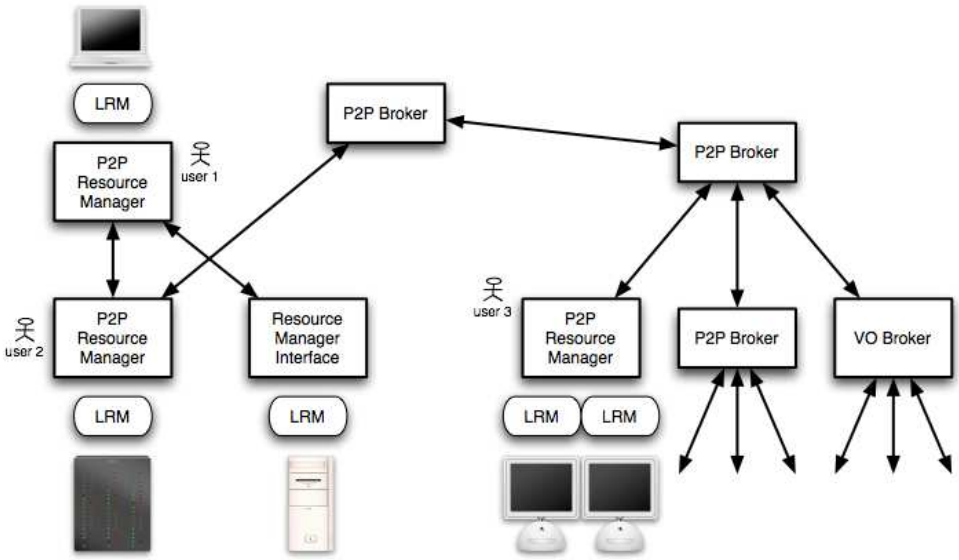

Figure 3: Example of a scheduling infrastructure for Global Grids

2 Scheduling Instances

It is possible to consider the different blocks in the previous examples as particular implementations of a more general software entity called scheduling instance. In this context, a scheduling instance is defined as a software entity that exhibits a standardised behaviour with respect to the interactions with other software entities (which may be part of a GSA implementation or external services). Such scheduling entities cooperate to provide, if possible, a solution to scheduling problems submitted by users, e.g. the selection and planning of resource allocations for a job [3].

The scheduling instance is the basic building block of a scalable, modular architecture for scheduling tasks/jobs/-workflows/applications in Grids. Its main function is to find a solution to a scheduling problem that it receives via a generic input interface. To do so, the scheduling instance needs to interact with local resource management systems that typically control the access to the resources. If a scheduling instance can find a solution for a submitted scheduling problem, the generated schedule is returned via a generic output interface.

From the examples depicted above it is possible to derive a high level model of operations for a generic set of cooperating scheduling instances. To provide a solution to a scheduling problem, a scheduling instance can exploit several options:

- It can try to solve the whole problem by itself interacting with local resource managers that it is able to interact with.
- If it can partition the problem in several sub-problems, it can try to:
  1. solve some of the sub-problems, if possible,
  2. negotiate to forward the unsolved sub-problems to independent scheduling instances,
  3. wait for potential solutions coming from other scheduling instances, or
  4. aggregate localised solutions to find a global solution for the original problem.
- If it cannot partition the problem or cannot find a solution by aggregating sub-problem solutions, it has two options:
  1. it can report back that it cannot find a solution or
  2. it can

– negotiate to forward the whole problem to another, different scheduling instance or
– wait for a solution to be delivered by the instance the problem has been forwarded to.

A generic Grid Scheduling Architecture will need to cover these behaviours, but actual implementations do not need to implement all of them. This model of operations is clearly modular, and permits to implement several scheduling infrastructures, like the ones depicted in the previous examples.

From them we can infer that a generic scheduling instance can exhibit the following abilities:

- interact with local resource managers;
- interact with external services that are not defined in the Grid Scheduling Architecture, like information, forecasting, submission, security or execution services;
- receive a scheduling problem (from other scheduling instances or external submission services), calculate a schedule, and return a scheduling decision (to the calling instance or an external service);
- split a problem in sub-problems, receive scheduling decisions and merge them into a new one;
- forward problems to other scheduling instances.

However, an instance might exhibit only a subset of such abilities. This depends on its interactions with other instances/services and its expected behaviour (e.g. the ability to split and/or forward problems).

If a scheduling instance is able to cooperate with other instances, it must exhibit the ability to send problems or sub-problems, depending on the case, and receive scheduling results. Looking at such an instance, we call higher level instances the ones that are able to directly forward a problem to that instance, and lower level instances the ones that are able to directly accept a problem from that instance. A single instance must act as a decoupling entity between the actions performed at higher and lower levels: it is concerned neither with the previous instances through which the problem flows (i.e. it has been submitted by an external service or forwarded by other instances as a whole problem or as a sub-problem), nor with the actions that the following instances will undertake to solve the problem. Every instance will need to know just the problem it has to solve and the source of the original scheduling problem that helps to resolve, to avoid potential forwarding issues.

From a component point of view abilities as described above are expressed as interfaces. In general, the interfaces of a scheduling instance can be divided in two main categories: functional interfaces and non-functional interfaces. The former are necessary to enable the main behaviours of the scheduling instance, while the latter are concerned with the management of the instance itself (creation, destruction, status notification, etc.). We want to highlight that we considered only the functionalities that must be directly exploited to support a general scheduling architecture; for example, security services are from a functional point of view not strictly needed to schedule a job, so they are considered external services or non-functional interfaces. The functional interfaces that a scheduling instance can expose are depicted in Figure 4 and in detail described in the following:

Input Scheduling Problems Interface The methods of this interface are responsible to receive a description of a scheduling problem that must be solved, and start the scheduling process. This interface is not intended to accept jobs directly from users; rather an external submission service (e.g. portal or command line interface) can collect the scheduling problems described with a user-defined formalism, validate them and produce a neutral representation accepted as input by this interface. In this way, this interface is fully decoupled from external interactions and can be exploited to compose several scheduling instance as in the examples illustrated above, where an instance can forward a problem or submit a sub-problem to other instances using this interface.

Every scheduling instance must implement this interface.

Output Scheduling Decisions Interface The methods of this interface are responsible to communicate the results of the scheduling process started earlier with a problem submission. Like the previous one, this interface is not intended to communicate the results directly to a user, rather to a visualization/reporting service. Again, we can exploit this decoupling in a modular way: if an instance received a submission from another one, it must use this interface to communicate the results to the submitting instance.

Every scheduling instance must implement this interface.

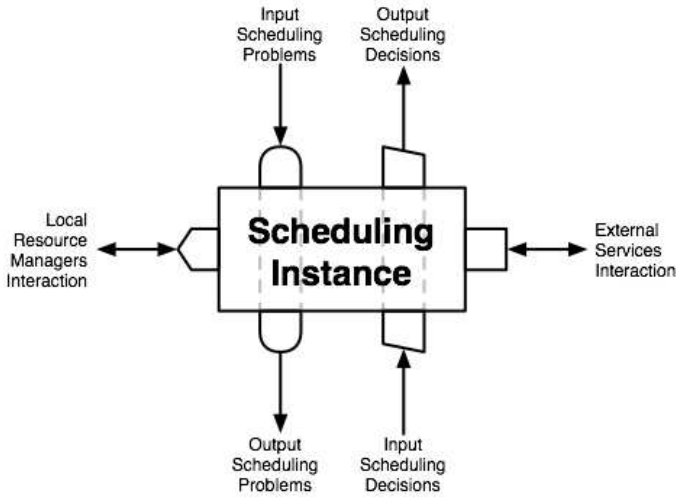

Figure 4: Functional interfaces of a scheduling instance

Output Scheduling Problems Interface If an instance is able to forward a whole problem or partial ones to other scheduling instances, it needs the methods of this interface to submit the problem to lower level instances.

Input Scheduling Decisions Interface If an instance is able to submit problems to other instances, it must wait until a scheduling decision is produced from the one which the problem was submitted to. The methods of this interface are responsible for the communication of the scheduling results from lower level instances.

Local Resource Managers Interface The final goal of a scheduling process is to find an allocation of the jobs to the resources. It means that sooner or later the whole process has to interact with local resource managers to allocate the jobs to the resources. While some scheduling instances can be dedicated to the "routing"of the problems, others interact directly with local resource managers to find suitable schedules, and propagate the answers in a neutral representation back to the entity that submitted the scheduling problem. Different local resource managers can require different interaction interfaces.

External Services Interaction Interfaces If an instance must interact with an entity that is neither a local resource manager nor another scheduling instance, it needs an interface that permits to communicate with that external service that is exploited by the scheduling architecture. For example, some instances may need to gain access to information, billing, security and/or performance predictor services.

Different external services can require different interaction interfaces.

It is not in the context of this work to define exactly the interfaces described above, their precise interactions and the relative communication protocols. However, the implementation of efficient Grid scheduling systems and strategies requires interaction with many Grid services that provide the core functionalities. Therefore, the existence of certain information, protocols, services and functionalities is required to build an efficient Grid scheduling architecture that allows the implementation of different Grid scheduling strategies.

3 Interactions between Scheduling Instances

While the interfaces provided by a scheduling instance are static, their state and behaviour are dynamic. They can depend on the identity of the user submitting the job, on the identity of the interacting scheduling instances, on local resource manager parameters. Therefore, every job submission must be followed by an exchange of information between several scheduling instances.

Every scheduling instance implements a set of strategies; the strategies must deal with user-dependent profit functions and local cost functions. During an interaction between higher-level and lower-level scheduling instances, the

information required from the former depends on the user-provided profit function and the implemented strategies, while the information required from the latter depends on its strategies and the cost function(s) of the underlying system(s).

For example, consider the Grid scheduling system depicted in Figure 5:

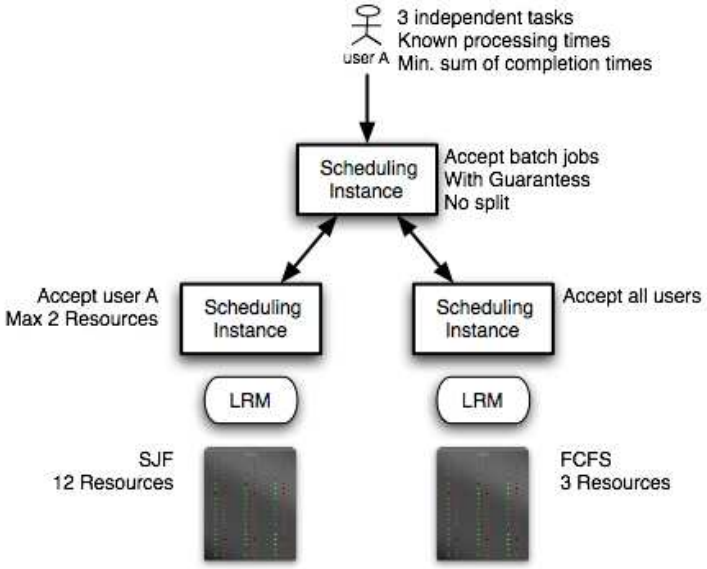

Figure 5: Example of Grid Scheduler strategies

If the user A submits the indicated job, the higher-level instance can ask for information about available time slots and will select the ones which minimise the total completion time (user profit function) without exploiting multi-site allocation (implemented strategy). On the other side, the lower-level instances will ask for identity of the submitting user (implemented strategy) and duration of the tasks (local cost function).

The interaction between scheduling instances requires exchange of information about scheduling strategies and job properties: a lower-level scheduling instance can ask for information about job characteristics, requirements and offers, while a higher-level scheduling instance can ask for planned schedules, characteristics of the underlying scheduling system and costs.

How the information is exchanged depends on negotiations procedures, both user- and system- dependent. When a schedule request is submitted to a Grid scheduling system, eventually it will reach a state where a higher-level scheduling instance, acting on the behalf of the user, will negotiate with a lower-level scheduling instance acting on the behalf of the underlying system(s). To this end, the higher-level instance uses his own or user policies to decide what kind of information is exposed to the lower-level system and how the negotiation is structured. Similarly, the higher-level system applies its policies to decide which offers are accepted or answered by counter-offers. On his side, the lower-level instance performs similar actions, exploiting his own and local system strategies.

After a setup phase, in which the strategies of the participants are inspected, the negotiation will start. At the end, an agreement between the instances might be reached. During the negotiation process, an agreement template is exchanged between the participants, which, in turn, fill it with the requested information.

It is possible to schematize the information that can be needed in the negotiation process in the following way:

User Provided: *Job Characteristics*: fixed attributes of the job independent from the scheduling system (e.g. processing times, time dependencies, single tasks profile, user authorization) that can be exploited in the negotiation process (e.g. to evaluate user profit function or system scheduling strategies).
Job Requirements: attributes that can need negotiation with the characteristics of the scheduling system to be arranged (e.g. target resource attributes, co-allocation of jobs, guarantees on job execution).
User Scheduling Objectives: mechanisms exploited by the higher-level scheduling instance to reach an agreement. It can include negotiation strategies and user profit functions.

System Provided: *Scheduling Characteristics*: fixed characteristics of the underlying scheduling system that can be exchanged during the negotiation (e.g. schedule, some allocation properties, system scheduling mechanisms to manipulate the allocation execution, etc.).
Allocation Requirements: attributes that can need negotiation with the higher-level scheduling instance and the user-provided information to be arranged (e.g. some allocation properties, job guarantees, etc).
System Scheduling Objectives: mechanisms exploited by the lower-level scheduling instance to reach an agreement. It can include negotiation strategies and system cost functions.

This information can be provided in several ways: for example (see Figure 6), the former one can be provided as an agreement template included in the job description submitted by the user, while the latter can be published (with a secure access mechanism, if needed) by the scheduling instance as a set of agreement templates containing a description of the job profiles accepted by the underlying system(s).

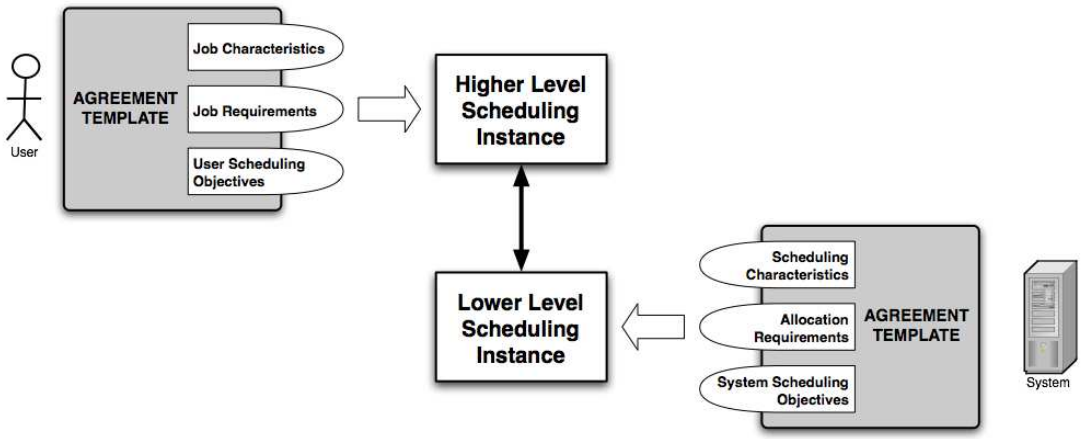

Figure 6: Information provided to Scheduling Instances

4 External Services Interactions

To solve scheduling problems, a GSA implementation can perform several tasks [4, 3]. To perform them, the scheduling instances composing the architecture can interact with several services, both external ones and those part of the GSA implementation. Such services must communicate using a specific protocol to convey information needed to solve the scheduling problems.

Exploiting the information presented in [1, 2], it is possible to identify a list of core, independent functions that can be used to build specific Grid scheduling systems. In the following a list of atomic, self-contained functions is presented; these functions can be part of any complex mechanism or process implemented in a generic GSA.

- **Naming**: Every entity in play must have a unique identifier for interaction and routing of messages. Some mechanism must be in charge of assigning and tracking unique identifiers to the involved entities.
- **Security**: Every interaction between different un-trusted entities can need several security mechanisms. A scheduling instance may need to certify its identity when contacting another scheduling instance, when it is trying to collect sensible information about other entities (e.g. planned schedules of other instances), or to discover what interactions it is authorized to initiate. Moreover, the information flow may need secure transport and data integrity guarantees, and a user may need to be authorized to submit a problem to a scheduling system. The security functions are orthogonal to other ones, in the sense that every service needs security-related mechanisms.
- **Problem Submission**: The entity implementing this function is responsible to receive a job to be scheduled from a user and submit it to a scheduling instance. At this level, the definition of job is intentionally vague, because it depends on the particular job submitted (e.g. a bag of tasks, a single executable, a workflow, a DAG). The job to be scheduled is provided using a user-defined language, and must be translated into a common description that is shared by some scheduling instances. This description will therefore be exploited in the whole scheduling process. It should be able to identify scheduling related terms and to build agreement templates used by the scheduling instances to schedule the job.
- **Schedule Report**: This function is the reciprocal of the previous one; an entity implementing it must receive the answer of the scheduling instance to a previously submitted problem and translate it into a representation consumable by the user.
- **Information**: A scheduling instance must have coherent access to static and dynamic information about resources characteristics (computational, data, networks, etc.), resource usage records, job characteristics, and, in general, services involved in the scheduling process. Moreover, it must be able to publish and update its own static and dynamic attributes to make them available to other scheduling instances. These attributes include allocation properties, local scheduling strategies, negotiation mechanism, local agreement templates and resource information relevant to the scheduling process [3]. It can be in addition useful to provide the capability to cache historical information.
- **Search**: This function can be exploited to perform optimized information gathering on resources. For example, in large scale Grids it can be neither important nor efficient to collect information about every resource, but just a subset of "good" candidate resources. Several search strategies can be implemented (e.g. "best fit" searches, P2P searches with caching, iterative searches). Every search should include at least two parameters: the number of records requested in the reply and a timeout for the search procedure.
- **Monitoring:** A scheduling instance can monitor different attributes to perform its functions: it can be useful to monitor e.g. the status of an agreement or an allocation to check if they are respected, the execution of a job to undertake next scheduling or corrective actions, or the status of a scheduling description through the whole system for user feedback.
- **Forecasting**: In order to calculate a schedule it can be useful to rely on forecasting services to predict the values of the quantities needed to apply a scheduling strategy. These forecasts can be based on historical records, actual and/or planned values.
- **Performance Evaluation**: The description of a job to be scheduled can miss some information needed by the system to apply a scheduling strategy. In this case it can be possible to exploit performance evaluation methodologies based on the available job description in order to predict the unknown information.
- **Reservation**: In order to schedule complex jobs as workflows and co-allocated tasks, as well as job with guarantees, it is in general necessary to reserve resources for particular time frames. The reservation of a resource can be obtained in several ways: automatically (because the local resource manager enforces it), on demand (only if explicitly requested from the user), etc. Moreover, the reservations can be restricted in time: for example only short-time reservations (i.e. with a finite time horizon) can be available. This function can require interaction with local resource managers and can be in charge of keeping information about allotted reservation and reserve new time frames on the resource(s).

- **Coallocation**: This function is in charge of the mechanisms needed to solve coallocation scheduling problems, in which strict constraints on the time frames of several reservations must be respected (e.g. the execution at the same time of two highly interacting tasks). It can rely on a low-level clock synchronization mechanism.
- **Planning**: When dealing with complex jobs (e.g. workflows) that need time-dependent access to and coordination of several objects like executables, data and network paths, a planning functionality, potentially built on top of a reservation service, is required.
- **Agreement**: In case quality of service guarantees concerning e.g. the allocation and execution time of a job must be considered, an agreement can be created and manipulated (e.g. accepted, rejected and modified) by the participating entities. A local resource manager can publish through its scheduling instance an agreement template regarding the jobs it can execute and a problem can include an agreement template regarding the guarantees that it is looking for.
- **Negotiation**: To reach an agreement the interacting partners may need to follow particular rules to exchange partial agreements to reach a final decision (e.g. who is in charge of providing the initial agreement template, who may modify what, etc.). This function should include a standard mechanism to implement several negotiation rules.
- **Execution**: This function is responsible to actually execute the scheduled jobs. It must interact with the local resource manager to perform the actions needed to run all the components of a job (e.g. staging, activation, execution, clean up). Usually it interacts with a monitoring system to control the status of the execution.
- **Banking**: The accounting/billing functionalities are performed by a banking system. It must provide interfaces to access accounting information, charging (in case of reservations or use of resources) and refunding (in case of agreement failures).
- **Translation**: The interaction with several services that can be implemented differently can force to translate information about the problem from the semantics of one system to the semantics of the other.
- **Data Management Access**: Data transfers can be included in the description of jobs. Although data management scheduling shows several similarities with job scheduling, it is considered a distinct, stand-alone functionality because the former shows significant differences compared to the latter (e.g. replica management and repository information) [5]. The implementation of a scheduling system can need access to data management facilities to program data transfers with respect to planned job allocations, data availability and eligible costs. This functionality can rely on previously mentioned ones, like information management, search, agreement and negotiation.
- **Network Management Access**: Data transfers as well as job interactions can need particular network resources to respect guarantees on their execution. As in the previous case, due to its nature and complexity, network management is considered a stand-alone functionality that should be exploited by scheduling systems if needed [6, 7]. This functionality can rely on previously mentioned ones, like information management, search, agreement and negotiation.

References

[1] R. Yahyapour and Ph. Wieder (Eds.). Grid Scheduling Use Cases v1.2. GGF-GSA Working Draft, 2005.

[2] U. Schwiegelshohn, R. Yahyapour, and Ph. Wieder. Resource management for future generation grids. Technical Report TR-0005, Institute on Scheduling and Resource Management, CoreGRID - Network of Excellence, May 2005.

[3] U. Schwiegelshohn and R. Yahyapour. Attributes for Communication between Scheduling Instances. GGF Document Series (GFD.6), 2001.

[4] J. M. Schopf. Ten Actions When Superscheduling. GGF Document Series (GFD.4), 2001.

[5] R. W. Moore. Operations for Access, Management, and Transport at Remote Sites. GGF Document Series (GFD.46), 2005.

[6] V. Sander (Ed.). Networking Issues for Grid Infrastructure. GGF Document Series (GFD.37), 2004.

[7] D. Simeonidou and R. Nejabati (Eds.). Optical Network Infrastructure for Grid. GGF Document Series (GFD.36), 2004.