Extension and Re-design of the eNANOS Grid Resource Broker for Supporting Grid Interoperability

Ivan Rodero and Julita Corbalan

Computer Architecture Department
Technical University of Catalonia (UPC)
Jordi Girona 1-3, Modul D6, 08034 Barcelona, Spain
{irodero,juli}@ac.upc.edu

Abstract. The grid resource management research has achieved several advances in last years. The interoperability among different grid systems, Service Level Agreements and job self scheduling are some of the current research topics. In this paper we present the new design and implementation of the eNANOS Grid Resource Broker that we have developed to support these approaches. In particular, the new design is based on services and XML schemas, and it is built on top of Globus Toolkit 4. Furthermore, we address the problem of the grid interoperability; first with our approach based on agreements, and afterwards we present our efforts on integrating the grid interoperability support into eNANOS within the LA Grid framework.

Key words: Grid Computing, Grid Resource Management, Grid Scheduling, Grid Broker, Grid Interoperability

1 Introduction

In the recent years a lot of efforts have been devoted in the research of grid resource management. In 2005 we presented the eNANOS Grid Resource Broker [11] that implemented interaction mechanisms with both the low-level support for High Performance Computing (HPC) systems [13], and the support for different Globus versions. It was built on top of Globus Toolkit 3 (GT3) infrastructure when GT3 and Open Grid Services Architecture (OGSA) were incoming approaches. The eNANOS Grid Resource Broker was fully exploited in the HPC-Europa project [15]. Meanwhile, several approaches concerning grid resource brokers have been developed in different contexts with different objectives. A complete and summarized taxonomy of the most important brokering approaches was presented in [5].

Currently, new paradigms have appeared such as the interoperability between grid systems, the job self scheduling or the Service Level Agreements. The required tasks to support interoperability among different brokers are many. Some of these tasks are cited in [12], including the relationship and capability management of grid brokers, job lifecycle management, resource capability, usage management, and so on. Different architectures have been proposed for grid interoperable systems, including HPC-Europa SPA [15], Gridway [4], Koala [8] and the OGF Grid Scheduling Architecture Research Group (GSA-RG) [10].

We started extending our framework to support these new functionalities. However, since eNANOS was built on top of GT3 that has become obsolete and unsupported, we decided to redesign the eNANOS framework and implement it on top of the Globus Toolkit 4 (GT4) infrastructure because GT4 is well- supported, and it is based on the Web Service Resource Framework (WSRF). Moreover, WSRF provides us a very good framework for a new component-based implementation using web services and its own persistency mechanism. We believe this framework will provide us the possibility to research in the current and incoming research topics such as the grid interoperability.

In this paper we present the new design and implementation of the eNANOS Grid Resource Broker. The main new design requirements are the modularity, extensibility, and service-based architecture. Since we are mainly focused on grid interoperability, we also present our first approach based on agreements, and afterwards we present our efforts on integrating the grid interoperability support in the eNANOS project within the LA Grid framework.

In section 2 we present some related work. In section 3 we present the new architecture and implementation of the broker. In section 4 we discuss our approach on grid interoperability and in section 5 the integration of eNANOS into the LA Grid framework. Finally, in section 6, we conclude the paper and present our future work.

2 Related Work

The need for interoperability among different grid systems in different domains was studied previously, and some projects have addressed this topic such as GRIP [3] and HPC-Europa SPA [15]. Lately, some initiatives have been started working on grid interoperability following similar objectives but in different directions. Gridway has incorporated the support for multiple grids in the last release. Its architecture consists of different categories, described in [4]. In the multiple meta-scheduler layers, Gridway instances can communicate and interact through called grid gateways, which can access resources belonging to different domains. The basic idea is to forward user requests to another domain when the current one is overloaded. Gridway is based on Globus, and they are experimenting with GT4 and gLite 3.

The Koala grid scheduler is another initiative, which is focused on data and processor co-allocation [8]. It was designed to work on DAS-2 multi-cluster and lately on DAS-3 and Grid5000. To inter-connect these different grid domains, they use inter-broker communication between different koala instances. Their policy is to use a remote domain only if the local one is saturated. They use delegated matchmaking where Koala instances delegate resource information using a P2P manner.

The GSA-RG of Open Grid Forum (OGF) [10] is currently working on enabling the grid scheduler interaction. They are working to define a common protocol and interface among schedulers enabling inter-grid resource usage, using standard tools (JSDL, OGSA, WS-Agreement). However, the group is paying more attention to agreements. In [14], they propose using WS-Agreement to make negotiations and perform interaction. They proposed the Scheduling Description Language (SDL) to allow specification of scheduling policies based on broker scheduling objectives (such as time constraints, job dependencies, scheduling objectives, preferences, etc.).

3 The New Design and Implementation

The key point of the new broker is its design based on services on top of GT4 and XML schemas. We have chosen GT4 instead of regular web services because it incorporates a persistency mechanism (Resource Properties), it is well-supported and stable. Moreover, WSRF provides us a very good framework to achieve some design requirements: modularity, extensibility, component-based, and easy deployment.

Our design objective is to provide a framework to deploy plug-ins on run-time that implements the required functionalities of the grid system. However, in our first implementation we keep the essence and mechanisms of the previous eNANOS Grid broker; for example the low-level support [13] or the scheduling performed in phases [11].

The overall architecture of the new eNANOS Broker version is presented in Figure 1. The core of the broker is composed by a set of services that can be deployed potentially in any machine with a GT4 container. The plug-ins are also implemented as GT4 services and provide access to the middleware or to external services. Since each core service is able to use different plug-ins at the same time, the broker can use different middleware or different middleware versions simultaneously. For example, the *"Dispatching"* Service can support different Globus versions or even Unicore, but it depends on the broker configuration and the plug-ins implementation.

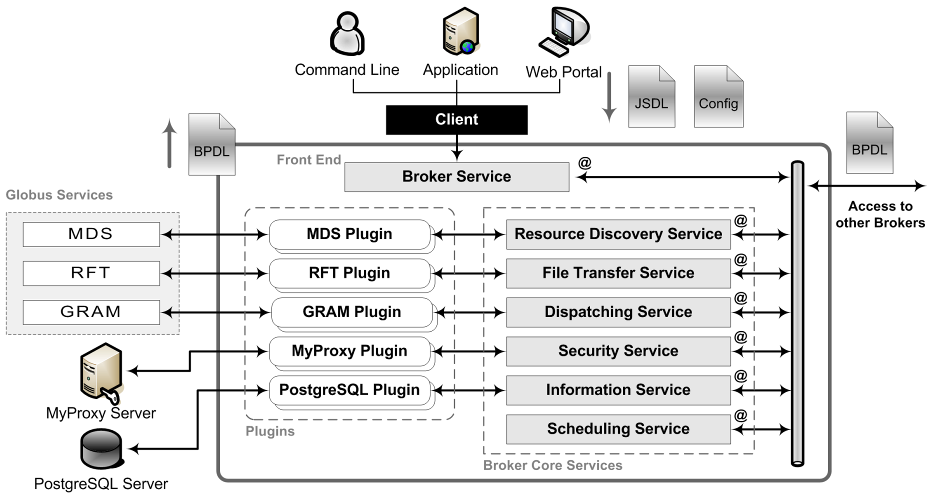

Fig. 1: Overall architecture of the new eNANOS Grid Broker

The *"Broker Service"* is the unique point of access to broker services and implements itself the full broker API. The rest of the services provides access to the plug-ins implementation through abstract services, and implement some logic depending of the service purpose. For example, the *"Scheduling Service"* collects queued jobs from the *"Information Service"*, available resources from the *"Resource Discovery Service"*, and it invokes the *evaluate* method of a specific policy plug-in through an abstract policy service. The *"Resource Discovery Service"* is responsible for obtaining the resources that are under the broker domain. For describing the resources we use a XML schema which is a subset of the Globus MDS (Monitoring and Discovering System) schema. In the current implementation we have a plug-in for Globus MDS and another one, called *custom plug-in*, that return a manually configured set of resources. This second plug-in is very useful for evaluating scheduling policies in a virtual scenario. The *"Information Service"* stores the information concerning the broker elements, such as jobs. It is able to provide both current and historical information. The first and simple plug-in was implemented using GT4 resource properties, but a powerful one based on postgreSQL database technology is under development. The *"Security Service"* manages the user credentials. In the current implementation we use X.509 proxy certificates and a plug-in based on Myproxy Server technology to store and to manage them. The *"Scheduling Service"* performs the scheduling of the queued jobs into the available resources. After collecting jobs and resources information, it invokes a policy plug-in through an abstract policy service. This mechanism allows including new scheduling policies just implementing a new plug-in and updating the broker configuration. Finally, the *"Dispatching Service"* is responsible for dispatching the job on a resource using a specific middleware. In the current implementation the monitoring mechanisms has been implemented as part of this service but we are working on a separate monitoring service. Other minor services are not shown in Figure 1 such as the *"JobIdFactory Service"* that provides unique Job IDs.

Since the presented design is very modular and decoupled, we can distribute the broker services into different machines and deploy them dynamically updating the services address (URI) in the broker configuration. For deploying a new service it only requires to follow the API of the specific service or plug-in, with no need to know the broker internals. For example, in the *"Scheduling Service"* we are able to get the available resources just with the address of the *"Resource Discovery Service"* and its API. From the point of view of the client side, the *"Broker Service"* is the unique point of access to the broker functionalities. It implements an API that can be used by different kind of clients such as web portals, user applications or command-line. For describing the jobs we use Job Submission Description Language (JSDL) with the POSIX extension [1].

As is shown in Figure 1, we also provide to the user the broker properties expressed in Broker Properties Description Language (BPDL) [7] [6], which provides a powerful schema to specify broker capabilities. Moreover, when different brokers are available in a grid system this information can be useful for selecting the most appropriate one. As well as providing a subset of the BPDL schema as monitoring information, we have defined a schema with the attributes that defines the configuration files of the broker. Basically, this schema includes the address and details of the broker services and plug-ins, and some other additional information. Moreover, we have defined schemas for describing jobs, resources, user configuration, and filters for querying jobs. These schemas are completely extensible for future functionalities or upgrades.

4 Interoperability Support

One of the main motivations of grid interoperability is to enlarge the grid domains using the resources and services of other grid domains, increasing, for example, the computational power or the total storage space. For the interoperability support, in our first approach we use the concept of agreement at the grid domain represented by a broker. The idea is obtaining the available resources from different brokers (potentially from different VOs) during the resource selection phase. If the most appropriate resource to submit a job is managed by another broker, we forward this job to this other broker. Figure 2 shows an example in which a set of brokers have established agreements between them.

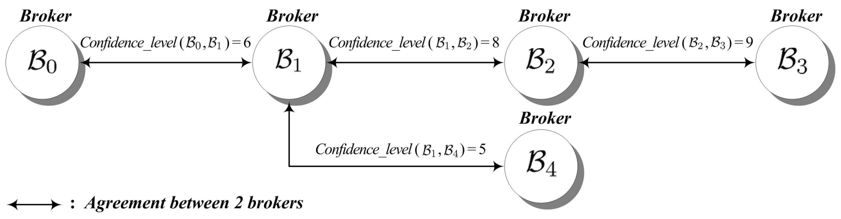
Broker
Broker
Broker
Broker
Broker
$\mathcal{B}_0$
$\mathcal{B}_1$
$\mathcal{B}_2$
$\mathcal{B}_3$
$\mathcal{B}_4$
Confidence_level ($\mathcal{B}_0$, $\mathcal{B}_1$) = 6
Confidence_level ($\mathcal{B}_1$, $\mathcal{B}_2$) = 8
Confidence_level ($\mathcal{B}_2$, $\mathcal{B}_3$) = 9
Confidence_level ($\mathcal{B}_1$, $\mathcal{B}_4$) = 5
: Agreement between 2 brokers

Fig. 2: Scenario example of broker aggreements

Considering that one broker ($\mathcal{B}$) manages the $\mathcal{R}$ set of resources, and another broker ($\mathcal{B}'$) manages the $\mathcal{R}'$ set of resources, $\mathcal{B}$ and $\mathcal{B}'$ can share their resources if they have an agreement between them. We define:

$$Agreement(\mathcal{B},\mathcal{B}') = true \Rightarrow \mathcal{B} \text{ and } \mathcal{B}' \text{ share their resources}$$

We assume that the agreements between brokers can be established indirectly. Thus, since two brokers with no direct agreement may share their resources, we use a confidence level when establishing agreements. In figure 2 we give a confidence level to each agreement, for example, between $\mathcal{B}_0$ and $\mathcal{B}_1$ the confidence level is 6. We consider that two brokers can share resources when they have a direct or an indirect agreement and the confidence level is greater than a given threshold (e.g. 5 in a scale from 1 to 10 like in the example of figure 2). Then, we define this property as follows:

$$if \;\; Agreement(\mathcal{B},\mathcal{B}') = true \;\wedge\; Agreement(\mathcal{B}',\mathcal{B}'') = true \;:$$

$$\begin{aligned} Agreement(\mathcal{B},\mathcal{B}'') = true \;\; \Leftrightarrow \;\; & Confidence\_level(\mathcal{B},\mathcal{B}') > threshold \\ \wedge \;\; & Confidence\_level(\mathcal{B}',\mathcal{B}'') > threshold \end{aligned}$$

For scalability and efficiency reasons we limit the number of indirect agreements between brokers. In particular, we scale the confidence level to the number of total agreement transitions between the brokers. Using this mechanism we reduce the resource domain obtained by agreements and we avoid cycles, because cycles in the agreements are very difficult to be managed in a real scenario.

$$
\begin{aligned}
&if \;\; Agreement(\mathcal{B}_0, \mathcal{B}_1) = true \;\; \wedge \ldots \wedge \;\; Agreement(\mathcal{B}_{n-1}, \mathcal{B}_n) = true \\
&\qquad Agreement(\mathcal{B}_0, \mathcal{B}_n) = true \;\; \Leftrightarrow \;\; \tfrac{Confidence\_level(\mathcal{B}_0, \mathcal{B}_1)}{n} > threshold \\
&\qquad\qquad \wedge \ldots \wedge \;\; \tfrac{Confidence\_level(\mathcal{B}_{n-1}, \mathcal{B}_n)}{n} > threshold
\end{aligned}
$$

The previous expression means that the $\mathcal{B}_0$ Broker considers that can use the $\mathcal{B}_n$ Broker resources. Taking the example of Figure 2 and considering a threshold of 4, $\mathcal{B}_0$ shares resources with $\mathcal{B}_1$, $\mathcal{B}_1$ shares resources with the rest of brokers, $\mathcal{B}_2$ shares resources with $\mathcal{B}_1$ and $\mathcal{B}_3$, $\mathcal{B}_3$ shares resources with $\mathcal{B}_1$ and $\mathcal{B}_2$, and $\mathcal{B}_4$ shares resources with $\mathcal{B}_1$.

Given the previous definitions, we consider that the typical scheduling process performed by a grid broker is divided into two phases: the job selection following a particular policy (e.g. FCFS, backfilling, etc.), and the resource selection depending on the job requirements. The typical resource selection performed by the $\mathcal{B}$ grid broker for a given job can be defined as follows.

$$
\begin{gathered}
Resources(\mathcal{B}) = \mathcal{R} \;\; \Leftrightarrow \;\; \forall r \in \mathcal{R} \; : \; r \;\; is\ managed\ by\ broker\ \mathcal{B} \\
Resource\_Selection(in\ job,\ in\ Resources(\mathcal{B})) \;\; \rightarrow \;\; out\ resource
\end{gathered}
$$

With the notion of agreements we can redefine the resource selection as follows:

$$
\begin{aligned}
&\qquad\qquad Resources'(\mathcal{B}) = \mathcal{R} \;\; \Leftrightarrow \;\; \forall r \in \mathcal{R} \; : \\
&r \;\; is\ managed\ by\ broker\ \mathcal{B} \vee (r \;\; is\ managed\ by\ broker\ \mathcal{B}' \wedge Agreement(\mathcal{B}, \mathcal{B}') = true)
\end{aligned}
$$

$$
Resource\_Selection'(in\ job,\ in\ Resources'(\mathcal{B})) \;\; \rightarrow \;\; out\ resource
$$

The resource selection selects the best resource following a particular policy. If the selected resource $r \notin Resources(\mathcal{B}) \;\wedge\; r \in Resources(\mathcal{B}')$, the job is forwarded to the $\mathcal{B}'$ Broker that manages the resource r.

For subsequent operations (e.g. canceling jobs or getting job status), we will forward the operations to the appropriate broker through a common interface. The implementation of a real interoperable grid system also requires other mechanisms such as a common resource model to exchange information between brokers, and an accounting mechanism when sharing resources.

5 Integration into LA Grid Project

Latin American Grid initiative (LA Grid, pronounced lah grid, [9]) is a multifaceted international academic and industry partnership designed to promote research, education and workforce development collaborations with major institutions in the United States, Mexico, Argentina, Spain, and other locations around the world. The meta-scheduling project in LA Grid aims to support grid applications with resources located and managed in different domains, spanned over a grid computing cyber infrastructure. This project addresses the architecture, design, implementation and deployment issues related to grid interoperability scheduling.

LA Grid meta-scheduling approach is very similar to our approach based on agreements. In LA Grid each domain is in charge of the resource publication and management in a Peer-To-Peer (P2P) fashion using the LA Grid meta-scheduler protocol [2]. Moreover, a set of interfaces and protocols have been defined for the interoperation. For establishing connection between two brokers, LA Grid uses a producer-consumer pattern; however, any broker can concurrently act in both roles on a single connection. Furthermore, a negotiation protocol is follow during the connection phase. Since in any reasonable sized grid the resource expression and exchange is a scalability issue, LA Grid uses an aggregated form of resource information exchanged among brokers (e.g., ProcType=(Intel, <count=5>) or FreeMem=(200-1500, <count=2>, <total=6000>)). The different institutions involved in LA Grid are able to interoperate with one another and submit jobs using JSDL.

Our implementation in LA Grid is based on the eNANOS version that is presented in this paper. The extended architecture of eNANOS is shown in figure 3 with the LA Grid extensions in dark shading. The extensions include a new service called *"LA Grid Service"* which is also a GT4 service. Since other LA Grid brokers have implemented the protocols using regular web services, eNANOS extensions have been implemented as a set of Axis2 services to avoid incompatibility problems with GT4 services. Some of the compatibility problems include the SOAP message formats and data types. The Axis2 services implemented on the server side acts as a wrapper to support re-directing calls to GT4 and performing data transformations when necessary. To support interactions between the eNANOS and other LA Grid brokers, we implement a set of regular web services for the APIs as the client interface. Moreover, in order to integrate eNANOS into LA Grid infrastructure we had to perform some modifications in the broker internals. In particular, we have updated the submission interface to support new parameters (such as the connection ID or the notification End-Point Reference), the job schema to manage forwarded jobs, the monitoring functionality to notify job changes to LA Grid infrastructure.

Significant modifications in the scheduling service are necessary to implement the scheduling policies based on aggregated resource information and to allow job forwarding. Thus, we implement a separate scheduling service for LA Grid because the changes are structural upgrades that break the consistency with other existing services. Moreover, we implement a scheduling policy for LA Grid using a new scheduling policy plug-in called “ *LAGridPolicyService*”. Since both services and plug-ins are implemented as modules with abstract interfaces, the upgrading of the broker was done by defining those new services and updating the configuration file. Moreover, we have implemented a new scheduling policy plug-in called “*BestBrokerRank*” policy. This policy selects the best broker to submit a job given a set of aggregated resource information. In case that the job is under the eNANOS domain, the policy returns the resource to execute the job and otherwise, it returns the broker to forward the job.

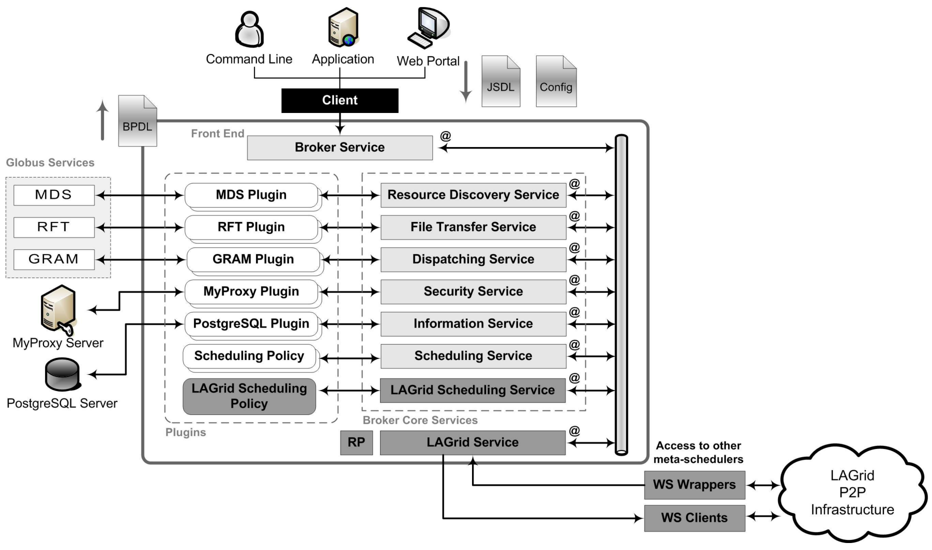

Fig. 3: Overall architecture the eNANOS grid broker with LA Grid extensions

Leveraging the default persistency mechanism of GT4, the data relevant to LA Grid functions is stored using the GT4 Resource Properties. The data to be stored include broker connections and resource information from other domains. We implemented a new set of resource management functions to support resource information exchange between partnering brokers. After obtaining the resource information within the domain, we create the aggregated form of resource information using the main attributes of resources and clustering the data by CPU and OS types.

In addition to receiving jobs routed from other LA Grid brokers through the *Job Management* API, the eNANOS broker can receive job submissions and other requests from regular users through the eNANOS clients that may be a command-line or a Java API (which can be used, for example, by a web portal or an external application), as shown at the top of figure 3. To allow eNANOS services to manage the LA Grid forwarded jobs, we modified the job schema used in eNANOS. In particular, we added a new element that contains a set of LA Grid information, and a new job status value (FORWARDED) for the jobs that have been forwarded to or from another broker domain. The extension of the schema was done through the XML schema type of figure 4, where *ConnID* is the connection ID, the *OriginatorEPR* is the end-point reference of the forwarding broker or the broker that has to receive notifications, and the job *realStatus* is the job status from a real execution environment.

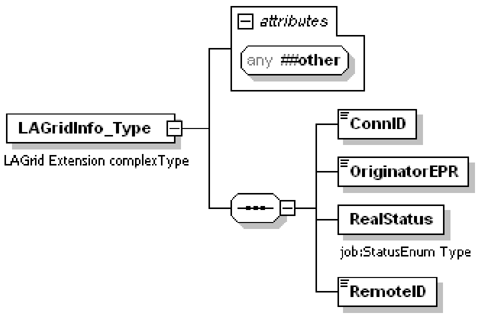

Fig. 4: Extended job schema of eNANOS

6 Conclusions and Future Work

In this paper we have presented the new design and implementation of the eNANOS Grid Resource Broker which is based on services and built on top of GT4. We also have discussed the grid interoperability problem as a one of the current research topics on grid resource management. Moreover, we have presented how we have addressed the problem of the grid interoperability: first with our approach based on agreements, and afterwards within the LA Grid framework.

Currently we are mainly focused on improving some broker details, developing new plug-ins, and incorporating new functionalities. The overall structure and mechanisms of the new eNANOS Grid broker are fully implemented but we need to work deeply in some services details, such as the implementation of the “Information Service” persistency through a PostgreSQL database which is under development. We also plan to implement a new “Cache Service” to improve the performance of the Resource Discovery functionality, and to update some logics depending on the system evaluation during the refinement process.

Concerning the grid interoperability support in eNANOS, we plan to continue the implementation of the LA Grid infrastructure and to continue our collaboration with STAKI [7] regarding the broker properties description language. In particular, we will research and implement policies for the selection of the most appropriate broker for a given job, depending on its resource requirements and the broker properties.

Acknowedgements

This paper has been supported by the Spanish Ministry of Science and Education under contract TIN2007-60625.

References

1. A. Anjomshoaa, M. Drescher, et al, Job Submission Description Language (JSDL) Specification Version 1.0, GFD-R.056, Tech. rep., Open Grid Forum (OGF) (November 2005).
2. N. Bobroff, L. Fong, Y. Liu, J. Martinez, I. Rodero, S. Sadjadi, D. Villegas, Enabling Interoperability among Meta-Schedulers, in: IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Lyon, France, 2008, pp. 306–315.
3. J. Brooke, D. Fellows, K. Garwood, C. Goble, Semantic Matching of Grid Resource Descriptions, in: European Acrossgrids Conference, Nicosia, Greece, 2004, pp. 240–249.
4. GridWay Project.
   `http://www.gridway.org`
5. A. Kertesz, P. Kacsuk, A Taxonomy of Grid Resource Brokers, in: Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS) in conjunction with the Austrian Grid Symposium 2006, vol. 32, Innsbruck, Austria, 2006, pp. 201–210.
6. A. Kertesz, P. Kacsuk, I. Rodero, F. Guim, J. Corbalan, Meta-Brokering Requirements and Research Directions in State-of-the-art Grid Resource Management, TR-0116, Tech. rep., Institute on Resource Management and Scheduling (2007).
7. A. Kertesz, I. Rodero, F. Guim, BPDL: A Data Model for Grid Resource Broker Capabilities, TR-0074, Tech. rep., Institute on Resource Management and Scheduling (2007).
8. KOALA Co-Allocating Grid Scheduler.
   `http://www.st.ewi.tudelft.nl/koala/`
9. Latin American Grid (LA Grid).
   `http://latinamericangrid.org/`
10. OGF Grid Scheduling Architecture Research Group (GSA-RG).
    `https://forge.gridforum.org/sf/projects/gsa-rg`
11. I. Rodero, J. Corbalan, R. Badia, J. Labarta, eNANOS Grid Resource Broker, in: European Grid Conference 2005, Amsterdam, 2005, pp. 111–121, LNCS 3470.
12. I. Rodero, F. Guim, J. Corbalan, L. Fong, Y. Liu, S. Sadjadi, Looking for an Evolution of Grid Scheduling: Meta-brokering, Grid Middleware and Services: Challenges and Solutions (2008) 105–119.
13. I. Rodero, F. Guim, J. Corbalan, J. Labarta, eNANOS: Coordinated Scheduling in Grid Environments, in: International Conference on Parallel Computing (ParCo), Malaga, Spain, 2005, pp. 81–88.
14. J. Seidel, O. Waldrich, W. Ziegler, P. Wieder, R. Yahyapour, Using SLA for Resource Management and Scheduling - a Survey, TR-0096, Tech. rep., Institute on Resource Management and Scheduling (2007).
15. HPC-Europa Project.
    `http://www.hpc-europa.org`