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

IANOS: An Intelligent Application Oriented Scheduling Middleware for a HPC Grid

Hassan Rasheed, Ralf Gruber, Vincent Keller
{Hassan.Rasheed, Ralf.Gruber, Vincent.Keller}@epfl.ch
EPFL
Lausanne, Switzerland

Oliver Wäldrich, Wolfgang Ziegler
{oliver.waeldrich, wolfgang.ziegler}@scai.fraunhofer.de
Fraunhofer Institute SCAI, Department of Bioinformatics
53754 Sankt Augustin, Germany

Philipp Wieder
philipp.wieder@udo.edu
IT & Media Centre, University of Dortmund
44221 Dortmund, Germany

Pierre Kuonen
Pierre.Kuonen@eif.ch
EIF
1075 Fribourg, Switzerland

Marie-Christine Sawley, Sergio Maffioletti, Peter Kunszt
CSCS
6928 Manno, Switzerland

CoreGRID Technical Report
Number TR-0110
December 28, 2007

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

IANOS: An Intelligent Application Oriented Scheduling Middleware for a HPC Grid

Hassan Rasheed, Ralf Gruber, Vincent Keller
{Hassan.Rasheed, Ralf.Gruber, Vincent.Keller}@epfl.ch
EPFL
Lausanne, Switzerland

Oliver Wäldrich, Wolfgang Ziegler
{oliver.waeldrich, wolfgang.ziegler}@scai.fraunhofer.de
Fraunhofer Institute SCAI, Department of Bioinformatics
53754 Sankt Augustin, Germany

Philipp Wieder
philipp.wieder@udo.edu
IT & Media Centre, University of Dortmund
44221 Dortmund, Germany

Pierre Kuonen
Pierre.Kuonen@eif.ch
EIF
1075 Fribourg, Switzerland

Marie-Christine Sawley, Sergio Maffioletti, Peter Kunszt
CSCS
6928 Manno, Switzerland

CoreGRID TR-0110
December 28, 2007

Abstract

We present the architecture and implementation of the IANOS middleware that is a result of integration of the Intelligent Scheduling Service into the VIOLA meta-scheduling environment. The goal of the new, integrated environment is to provide a general middleware infrastructure allowing optimal positioning and scheduling of HPC Grid applications. The scheduling algorithms used to calculate best-suited resources are based on an objective cost function that exploits information on the parameterization of applications and resources. The overall middleware architecture comprises four general services and two modules. The implementation is based on state-of-the-art Grid and Web services technology as well as existing and emerging standards (WSRF, WS-Agreement, JSDL, and GLUE). IANOS in general is agnostic to a Grid middleware. Currently, the IANOS integration with the UNICORE Grid system is under development.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Introduction

The Intelligent Scheduling System (ISS) aims at finding optimally suited computational resources for a given application. It uses historical runtime data of an application to schedule well suited resources for execution based on performance requirements of the user. The scheduling algorithms used to calculate best-suited resources are based on a cost function that exploits information on the parameterization of application and resources. The collected data on Grid resources and the monitored data on past application executions can be used to detect overloaded resources and to pin-point inefficient applications that could be further optimized.

The VIOLA Meta-Scheduling Service (MSS) co-allocates computing as well as network resources in a coordinated fashion. The MSS is a Grid scheduler, receiving a list of resources and returning reservations for some or all of these resources. To achieve this, the MSS first queries selected local scheduling systems for the availability of these resources and then negotiates the reservations across all local scheduling systems with available resources. The communication between MSS and the involved RMSs can either be done directly or through a standard Grid middleware system. At the moment MSS supports UNICORE [9] Grid middleware. The integration into UNICORE 6 and Globus Toolkit 4 is under development.

The Intelligent ApplicatioN Oriented Scheduling (IANOS) scheduling middleware is a result of integration of ISS into MSS. The IANOS is a general scheduling middleware comprising four services and two modules: the *Meta Scheduling Service* (MSS) performs resource discovery and job submission; the *Grid Adapter* (GA) module is part of MSS which provides generic interfaces and components to interact with different Grid middlewares; the *Resource Broker* (RB) is responsible for the selection of suitable resources; the *System Information* (SI) is a front end to Data Warehouse, analyzes the stored data and compute certain free parameters to be used by ISS models; the *Monitoring Service* (MS) monitors the application during execution and computes execution relevant quantities; the *Data Warehouse* (DW) module is part of SI which stores information on applications, resources and execution related data; and a client which submits the application to MSS using WS-Agreement.

The implementation is based on state-of-the-art Grid and Web services technology as well as existing and emerging standards like Web Services resource Framework (WSRF) [12], WS-Agreement (WSAG) [2], Job Submission Descricption Language (JSDL) [3], and GLUE [1]. The IANOS middleware is general in the sense that it is agnostic to a Grid middleware and therefore, can be used with different Grid middlewares. For a first phase, the adapter for a UNICORE Grid middleware [9] has been partially implemented and the IANOS integration with UNICORE System is under development.

The first draft of the architecture, which has been presented in the CoreGRID technical report TR-0025 [4], has been refined during the implementation during the first phase of the project. The resulting architecture is included in this paper. Moreover, the CoreGRID TR-0070 [5] gives an deep insight into the underlying models for both capturing machine and application behaviour, which are hence only briefly sketched in this paper.

In section 2, a short description of ISS models is presented. The MSS is briefly discussed in section 3. The IANOS middleware architecture is detailed in section 4. The reference scheduling scenario is explained in section 5. Section 6 describes the integration in the UNICORE environment, and the last section provides a summary and information on future work.

2 Intelligent Scheduling Service

The ISS is supposed to help deciding on which Grid resources a scientific application should be executed. ISS has four models as fellows: Gamma model, Performance Prediction model, Execution Time Evaluation model, and Cost Function model. The *Gamma model* evaluates the suitability of resources for a given application. The *Performance Prediction model* predicts the performance of an application on a resource where the application never was executed, using results of the same application's run on another known resource. The *Execution Time Evaluation model* forecasts the execution time of a given application on a given resource. The *Cost Function model* calculates the cost value for each candidate resource. We refer to [7] [6] for in depth details on these models. The Grid resource parameters required to compute ISS models relevant quantities are shown in Tables 1 and 2.

2.1 Assumptions

- The user is able to provide some application relevant intrinsic parameters such as size of the problem, number of iterations, etc

- Applications are well balanced
- The computing nodes or machines of each Grid resource are homogeneous
- All Grid resources share their ISS relevant parameters as shown in Tables 1 and 2

2.2 Gamma Model

The Gamma model evaluates the suitability of resources for a given application. The Gamma model is defined as a ratio between computation and communication time of application on a given resource. From the parameters shown in Tables 1, we can compute the Gamma model relevant quantities, which are also used in other models. The Gamma model provides a good insight on the suitability of a given hardware to run an application efficiently.

2.3 Performance Prediction Model

The ISS decision process is based on a single metric: the COST which includes the cost of the execution of an application component on a Grid resource. We need to predict the performance of application on a potential unknown Grid resource. If one is able to predict the performance of an application on the new resource (where the application has never been executed) based on the performance of the application on a resource (on which the application has been executed), then we are able to predict the execution time and the costs of the run.

2.4 Execution Time Evaluation Model

There is an execution time cost as part of the cost function. The Execution Time Evaluation model estimate the execution time of a given application on a given resource. If one is able to estimate the execution time of an application on an unknown Grid resource based on the performance and on the time of the execution of the same application on a known resource, then one is able to predict the cost of this application on all the Grid resources.

Table 1: ISS relevent Quantities.

Resource Characteristics	Quantities collected during Application Execution
Number of nodes	Execution Time
Number of Processors Per Node	Average Efficiency
Number of Cores Per Processor	Average Performance
Peak CPU Performance	Number of Packets Sent (Average)
CPU Performance Factor	Number of Packets Received (Average)
Peak Main Memory Bandwidth	Sent Packet Size (Average)
Peak Network Bandwidth	Received Packet Size (Average)
Machine Architecture	Memory used in [Bytes]
Operating System	Swap used in [Bytes]
Network Topology	
Interconnect Network NIC's Type	

2.5 Cost Function Model

The cost function model depends on costs due to machine usage, license fees, energy consumption and cooling, waiting results time, and amount of data transferred. All these quantities depend on the application components, on the per hour costs of a resource with a total number of computational nodes, on the number of processors used in the computation for each component, and on data transfer costs over the Internet. We express the money quantity as Electronic Cost Unit ([ECU]). In summary, it depends on the following costs:

- CPU Costs
- Licensing Cost

- Waiting Time Cost
- Data Transfer Cost
- Energy Cost

In CF model, the four parameters α, β, γ, and δ are used to weight the user preferences. Because, some users would like to obtain the result of their application execution as soon as possible, regardless of costs; some others would like to obtain results for a given maximum cost, but in a reasonable time, and some others for a minimum cost, regardless of time. Therefore, the user can prescribe two constraints: Maximum Cost and Maximum Turnaround Time. These four parameters have to be tuned according to the policies of the computing centers and user's demands. In fact, the user's (resource consumer) and the computing center's (resource provider) interests are divergent. This implies a constant tuning of the free parameters to satisfy both somewhat contradictory goals.

Table 2: Cost Function Model Parameters.

Cost Function Model Parameters
Machine Online Time
Regression Factor (in [1/Year])
Amortizement Time (in years)
Host Efficiency (over the year)
Hours (non-bissextile year)
Node's Energy Consumption
The cost for a KWh
Investment Cost
Interest Cost
Personal Cost
Infrastructure Cost
Management Overhead
Insurance Fee Cost
Software licence Cost

3 Meta Scheduling Service

In the German VIOLA project [10], a Meta Scheduling Service (MSS) has been developed [11] to co-allocate network resources as well as computing resources. The MSS is a Grid scheduler capable of orchestrating and co-allocating resources. In order to co-allocate resources, MSS mediates between the RMS involved. During this process, the MSS creates resource reservations, which are coordinated in time.

The MSS offers on the one hand the support for workflows where the agreements about resource or service usage (aka reservations) of consecutive parts should be made in advance to avoid delay during the execution of the workflow. On the other hand the Meta Scheduling Service also supports co-allocation of resources or services in case it is required to run a parallel distributed application which needs several resources with different characteristics. The local schedulers are contacted via an adapter that provides a generic interface to these schedulers. The negotiation process between the Meta-Scheduling Service and the adapters is implemented based on a proprietary protocol.

The communication between MSS and the involved RMSs can either be done directly or through a standard Grid middleware system. At the moment the MSS supports UNICORE version 5. However, its design allows to support different Grid middlewares using Grid Adapters.

4 IANOS middleware Architecture

The following section presents finer details of services and modules of the middleware including some discussions on design considerations.

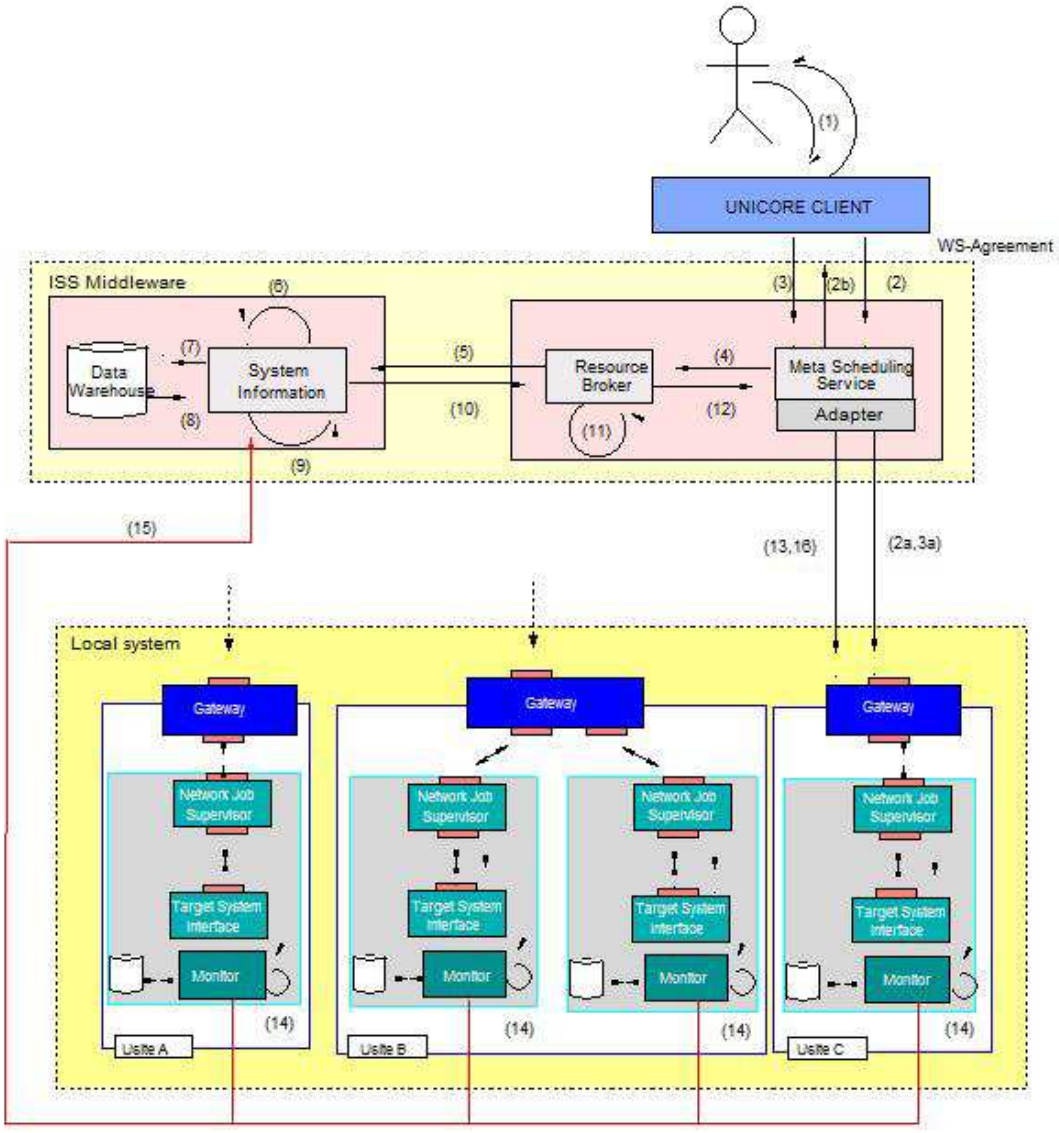

Figure 1: IANOS Middleware Architecture with UNICORE Grid System

4.1 Grid Adapter

The Grid Adapter mediates access to a Grid System through a generic set of components and interfaces. The SiteManager queries Grid system for a list of available Grid Sites. The ApplicationManager contacts available Grid Sites for their installed applications. Each application is modeled as a *WSAG-Template*. The InformationManager provides both static and dynamic information on the resources. It includes static information on hardware and software configuration, usage policies, and dynamic information on resource availability, i.e, current load. All this information is modeled as an *extended GLUE schema*. The DataManager is responsible for the staging in and out of job files. The JobManager is responsible for job submission and management. The ReservationManager handles the reservation of computational and network resources in advance. The resources are reserved for a certain time counting from the agreed start time. The JobManager and the DataManager receive *JSDL* as input while the ReservationManager receives *WS-Agreement* as an input. A minimal set of interfaces has been designed for each component for extensibility reasons.

4.2 Meta Scheduling Service

The MSS is the only IANOS service that is connected to a Grid system and is accessible by the client. It performs candidate resources selection; provides information on resources including CPU time availability, and handles reservation/submission of jobs on a selected resource. It uses the Grid Adapter to access the Grid system, and the access from the client is provided by an *AgreementFactory* interface. The client and the MSS communicate using the WS-Agreement protocol. The Client requests installed applications by issuing an *AgreementTemplate* request. The MSS first validates user requests and then queries the underlying Grid system for the list of installed applications based on user authorization filtering and application availability. Then the MSS sends the information on available applications to the client in the form of AgreementTemplates. The client selects and submits the application to MSS by sending an *AgreementOffer*. This AgreementOffer includes a job description, application parameters, and user QoS preference, e.g. maximum cost, maximum turnaround time.

Upon receiving an *AgreementOffer* from client, it first identifies potential candidate resources and then queries the Grid Adapter for each candidate resource's static and dynamic information, which is received in *GLUE* format. It also retrieves the CPU time availability information (TimeSlots) for each candidate resource by contacting their local RMS. The application is represented by an extended *JSDL* with information on instrinsic application parameters, and user QoS preferences, and *GLUE* contains the information on candidate resources. The MSS sends JSDL and GLUE documents to the Resource Broker.

The response from the Resource Broker is an ordered list of execution configurations. Each configuration is represented by a JSDL document and includes start and end time of the execution, required number of nodes, and cost value for this configuration. The MSS starts negotiation with resources and uses the GA to schedule one of the configurations following the preferences expressed in the ordered list. A *WS-Agreement Resource* is created for each successfully submitted job on a selected resource, with job information stored as resource properties. The MSS supports complete job management, such as job monitoring and control.

4.3 Resource Broker

The Resource Broker service exposes only one operation, *getSystemSelection*. The parameters of this operation are the *JSDL* and *GLUE* documents representing the candidate resources, the application parameters and the user QoS preference. The Resource uses four modules to decide on the suitable resources for a given application. The *GammaModule* implements Gamma model, the *EtemModule* implements Execution Time Evaluation model, the *GpemModule* implements Performance Prediction model, and the *CfmModule* implements Cost Function model. An interface is also designed for each module. This implementation framework allows separating implementation of different models and to extend or provide new implementations of the models.

To compute a cost value for each TimeSlot of the candidate resources, the RB needs data on ISS relevant resource parameters shown in Tables 1 and 2, and data on the given application characteristics and requirements. For this purpose, the RB contacts System Information for this data. The Resource Broker after receiving required information, filters out unsuitable candidate resources based on the application requirements (nodes, memory, libraries etc). It then evaluates the cost function values and tolerances for the different candidate resourcesand. It prepares an ordered list of suitable configurations (including start-time and deadlines) and sends them to the MSS. Each configuration along with job requirements is represented by a *JSDL document*.

4.4 System Information

It exposes three operations: *getAppISSInfo*, *updateAppExecutionInfo* and *updateISSInfo*. The *getAppISSInfo* operation provides data on given resources and applications to the Resource Broker. The *updateAppExecutionInfo* operation receives execution related data from a Monitoring service. The *updateISSInfo* operation receives ISS relevant static resource data from a Monitoring service.

The System Information is a frontend to the Data Warehouse module. An interface is designed to allow integration between System Information and Data Warehouse independent of specific implementation of the DW. The Data Warehouse is contacted by the System Information to query, add or update the data. The SI has another module to compute free parameters from previous execution data and application parameters.

4.5 Data Warehouse

The Data WareHouse is a repository of all information related to the applications, to the resources found, to the previous execution data, and to the monitoring after each execution. Specifically, the Data Warehouse contains the following information:

- Resources: Application independent hardware quantities, ISS related characteristic parameters
- Applications: Application characteristics and requirements (software, libraries, memory, performance)
- Execution: Execution data for each executed run (Execution dependent hardware quantities, application intrinsic parameters, free parameters)

A web interface is also provided to add/update some of this information into Data Warehouse.

4.6 Monitoring Service

The Monitoring service [8] has been developed to measure and collect ISS relevant execution quantities (MFLOPS/s rate, memory needs, cache misses, communication, network relevant information, etc) during application execution. These quantities can be computed using direct access to hardware counters using PAPI. It performs a mapping between hardware monitored data using the Ganglia service and application relevant data using the RMS (local scheduler Torque/Maui). At the end of the execution, the Monitoring Service prepares and sends monitored data to SI as shown on right side of Table 1.

5 Scheduling Scenario

Figure 1 presented in the previous section shows the integration of IANOS middleware with UNICORE. This section describes a reference scenario of job submission using IANOS middleware. The job submission process has been divided into 4 phases:

- Prologue (1-4)
- Decision (5-11)
- Submission (12-13)
- Epilogue (14-16)

Each flow of data is represented by a number and an arrow in Fig. 1. The reference scenario is presented in detail below.

1 User logs in to the client

2 User requests the list of installed applications from MSS by sending a WS-Agreement template request

$2a$ MSS validates and authenticates the client request, contacts GA for all Grid Sites, and then queries for the list of installed applications based on user authorization filtering

2b Prepare and return WS-Agreement templates (represent the applications) to the client

3 User selects one application, specifies application intrinsic parameters and QoS preference (Cost, Time, Optimal), and submits the application as an AgreementOffer to MSS

3a MSS validates the agreement offer, selects candidate resources based on the user access rights and the application availability, queries the Grid Adapter for each candidate resource's static and dynamic information including the resource's local RMS availability

4 MSS sends the candidate resources, the application intrinsic parameters and user QoS preference to RB

5 RB requests ISS relevant resource parameters and data on the given application characteristics and requirements

6 SI analyzes the request and prepares queries to retrieve the required data from DW

7 SI requests the required information from DW through DW interface

8 DW sends collected information to SI on the candidate resources, on the application, and on the previous execution data of application

9 SI computes Etem model's free parameters from previous execution data and application intrinsic parameters

10 SI sends requested data including the free parameters to RB

11 RB filters out unsuitable candidate resources based on the application requirements (nodes, memory, libraries etc). It then evaluates the cost function model and prepares a list of cost function values and tolerances for all candidate resources based on the user's QoS preference

12 RB selects an ordered list of suitable configurations after applying the cost function and sends them to the MSS

13 MSS uses GA to schedule one of the configurations following the preferences expressed in the ordered list, and WS-Agreement Resource is created for each scheduled configuration

14 MS monitors the execution of the application and saves the application's execution data in a local database

15 After the completion of the job, MS computes the execution relevant quantities and sends them to SI

16 At the end of execution, results are being sent to the client

6 Integration with UNICORE Grid System

In a first phase of the IANOS project, we have integrated the IANOS middleware with UNICORE Grid System. The Grid Adapter is the IANOS module which mediates access to Grid middleware functionality and services. Therefore, a first version of the Grid Adapter for UNICORE has been implemented. The three modules, SiteManager, InformationManager, and SubmissionManager have been developed while the DataManager and ReservationManager modules are currently under development.

On the client side, a UNICORE Client plugin has been developed for this purpose. This client plugin provides a GUI to gather the user's application specific and QoS related input and interacts with the MSS.

7 Conclusion and Future Work

We have presented the design and implementation of the IANOS middleware that is a general scheduling framework for HPC Grid applications. I is intended to be as Grid middleware independent as possible, and is based on (proposed) Grid and Web services standards. Currently the integration with the UNICORE system is under developement.

It is planned to release a beta version of ISS middleware in Feb, 2008. The resulting middleware will be validated on a large scale testbed by running different types of HPC applications. This testbed includes resources from four institutes: EPFL, Fraunhofer SCAI, CSCS (Manno) and University of Dortmund.

8 Acknowledgements

ISS is a Swiss project within the Swiss Grid Initiative SWING and is managed by the Swiss National Supercomputing Centre. MSS is funded by the German FederalMinistry of Education and Research through the VIOLA project under grant #01AK605L. The IANOS integration work is carried out jointly within the CoreGRID Network of Excellence funded by the European Commission IST programme under grant #004265.

References

[1] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. K'onya, M. Mambelli, J. M. Schopf, M. Viljoen, and A. Wilson. Glue schema specification version 1.2. Technical report. Web site. Last access 15. Dec 2007: <http://glueschema.forge.cnaf.infn.it/Spec/V12>.

[2] A. Andrieux et al. Web Services Agreement Specification (WS-Agreement). Grid Forum Document GFD.107, Open Grid Forum, 2007.

[3] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification v1.0. Grid Forum Document GFD.56, Global Grid Forum, November 2005.

[4] Kevin Cristiano, Pierre Kuonen, Ralf Gruber, Vincent Keller, Michela Spada, Trach-Minh Tran, Sergio Maffioletti, Nello Nellari, Marie-Christine Sawley, Oliver Wäldrich, Wolfgang Ziegler, and Philipp Wieder. Integration of ISS into the VIOLA Meta-Scheduling Environment. Technical Report TR-0025, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, January 2006.

[5] Alain Drotz, Ralf Gruber, Vincent Keller, Michela Thiemard, Ali Tolou, Trach-Minh Tran, Kevin Cristiano, Pierre Kuonen, Philipp Wieder, Oliver Wäldrich, Wolfgang Ziegler, Pierre Manneback, Uwe Schwiegelshohn, Ramin Yahyapour, Peter Kunszt, Sergio Maffioletti, Marie-Christine Sawley, and Christoph Witzig. Application-oriented scheduling for HPC Grids. Technical Report TR-0070, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, February 2007.

[6] R. Gruber, V. Keller, P. Kuonen, M.-C. Sawley, B. Schaeli, A. Tolou, M. Torruella, and T.-M. Tran. Towards an intelligent grid scheduling system. In *Proceedings of the 6th International Conference, Parallel Processing and Applied Mathematics, PPAM 2005,*, volume 3911 of *LNCS*, pages 751 – 757, Poznan, Poland, September 2005. Springer.

[7] R. Gruber, P. Volgers, A. De Vita, M. Stengel, and T.-M. Tran. Parameterisation to tailor commodity clusters to applications. *Future Generation Computer Systems*, 19:111–120, 2003.

[8] A. Keller. VAMOS web frontend to the Pleiades clusters. Web site. Last access 15 Dec 2007: <http://pleiades1.epfl.ch/ vkeller/VAMOS/>.

[9] Web Site. UNICORE Open Source Download , 2007. Last access 15 Dec 2007: <http://www.unicore.eu/download/unicore5/>.

[10] Web Site. VIOLA, Vertically Integrated Optical Testbed for Large Application in DFN, 2007. Last access 15 Dec 2007: <http://www.viola-testbed.de/>.

[11] Philipp Wieder, Oliver Wäldrich, and Wolfgang Ziegler. A meta-scheduling service for co-allocating arbitrary types of resources. In *Proceedings of the 6th International Conference, Parallel Processing and Applied Mathematics, PPAM 2005,*, volume 3911 of *LNCS*, pages 782 – 791, Poznan, Poland, September 2005. Springer.

[12] OASIS Web Services Resource Framework (WSRF) TC. 3 Apr 2006. Web site. Last access 15 Dec 2007: <http://www.oasis-open.org/committees/wsrf>.