TOWARDS A STANDARDS-BASED GRID SCHEDULING ARCHITECTURE

Christian Grimme, Joachim Lepping, Alexander Papaspyrou, Philipp Wieder, and Ramin Yahyapour
Dortmund Technical University, IRF & ITMC
44221 Dortmund, Germany
{christian.grimme, joachim.lepping, alexander.papaspyrou, philipp.wieder, ramin.yahyapour}@udo.edu

Ariel Oleksiak
Poznan Supercomputing and Networking Center
Noskowskiego 10 61-704, Poznan, Poland
ariel@man.poznan.pl

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

Abstract The definition of a generic Grid scheduling architecture is the concern of both the Open Grid Forum's Grid Scheduling Architecture Research Group and a CoreGRID research group of the same name. Such an architecture should provide a blueprint for Grid system and middleware designers and assist them in linking their scheduling requirements to diverse existing solutions and standards. Based on work executed within OGF related to scheduling use cases and requirements, which tackles the problem from a more theoretical point of view, we approach in this paper the problem practically by evaluating the *teikoku* scheduling framework in the light of standards-compliance. The results of this evaluation and the existing Grid Scheduling Architecture proposal are set into context, existing gaps are described and potential solutions to bridge them are introduced. In doing so, we concentrate on the interoperability of schedulers and the necessity of a Scheduling Description Language to achieve it.

Keywords: Grid Scheduling Architecture, Scheduling Description Language, Open Grid Forum

1. Motivation

The overall goal of our work is the provision of a generic Grid scheduling architecture. This work is primarily conducted by the *Grid Scheduling Architecture Research Group* (GSA-RG[1]) of the Open Grid Forum and a research group of the same name placed within CoreGRID's Institute on Scheduling and Resource Management[2]. The primary objectives defining the architecture are the following three:

1 **Standards-compliance**. The Grid scheduling architecture should, wherever possible, be based on languages, protocols, and specifications which are standards-compliant.

2 **Interoperability**. Implementations following the Grid scheduling architecture blueprint should be interoperable.

3 **Universality**. A realisation of the Grid scheduling architecture should be possible without following a specific design paradigm or using a certain technology.

We approach the problem space from two viewpoints, a generic (and more theoretical) one defining generic use cases and requirements, and from a practical one evaluating a standards-based Grid scheduler (cf. Section 2). As the former has already been described in a number of publications, we concentrate here on the latter by outlining the features, design, and architecture of the *teikoku Grid Scheduling Framework* (tGSF; cf. Section 3). Based on this framework and the research as well as production requirements set, we outline the gaps in the current standards landscape, reflect upon interoperability of Grid schedulers, and sketch the efforts necessary towards a standards-compliant, interoperable, and universal Grid scheduling architecture definition (cf. Section 4). In a concluding section we describe the efforts already under way and the steps to be taken in the near future.

Please note that we use the term "standards" in a wider sense throughout this paper. We do not merely refer to fully specified, tested, and certified standards, but also to specifications which are under development and are likely to become standards some time in the future.

2. Problem Space

The scheduling and allocation of tasks, jobs, or workflows on a set of heterogeneous resources in a dynamically changing environment is a

[1] `https://forge.gridforum.org/sf/projects/gsa-rg`
[2] `http://www.coregrid.net/mambo/content/blogcategory/16/295`

complex problem. There are still no common Grid scheduling strategies available which solve this problem and implementations of scheduling systems still require specific architectures customised for the target computing platform and the application scenarios. Moreover, the complexity of applications, variety of user requirements, and system heterogeneity do not permit the efficient manual performance of any scheduling procedure.

Although no common and generic Grid scheduler yet exists, several common aspects can be found examining existing Grid scheduling uses cases [13] and Grid schedulers [12]. This leads to the assumption that a generic architecture may be conceivable not only to simplify the implementation of different schedulers but also to provide an infrastructure that enables interoperability between those different systems.

Furthermore, a number of standards that are useful in a Grid scheduling context are already specified and are also implemented in a number of Grid middlewares and services. Such standards are – ideally – being developed by consortia comprising representatives from academia and industry. Regarding Grids, the respective consortium is the Open Grid Forum[3]. It has produced scheduling-related standards like the Job Submission Description Language (JSDL [2]) for job submission, the Web Services Agreements specification (WS-Agreement [1]) for definition and negotiation of Service Level Agreements, the OGSA Basic Execution Service (OGSA-BES [9]), which is used to initiate, monitor, and control computational activities, or the OGSA Resource Selection Services (OGSA-RSS[4]), which provide means to discover and selection resources.

Since we have already gained good understanding of common requirements regarding a Grid Scheduling Architecture, we decided within CoreGRID to tackle the problem also from a practical perspective in order to adjust our theoretical findings. Our candidate Grid scheduler, which is described in detail in the following section, has been designed and implemented with standards-compliance in mind. It is evaluated in the light of our architectural objectives (see Section 1) and the findings are related to the general Grid scheduling requirements.

3. A Standards-based Grid Scheduler

Although Grid scheduling has become an important topic in various businesses and many commercial Enterprise Grid solutions[5] are available at the market, the development of advanced scheduling strategies and

[3] `http://www.ogf.org`
[4] `https://forge.gridforum.org/sf/projects/ogsa-rss-wg`
[5] Such as LSF (`http://www.platform.com`).

algorithms is mainly a matter of scientific research. However, the transferability of the gained results into production environments is problematic at best, since model assumptions are usually not in line with the conditions in enterprise scenarios.

The teikoku Grid Scheduling Framework tries to bridge the gap between research and production systems, the requirements of which differ significantly:

- *Research Requirements* include the availability of an extensible toolbox that supports the development and analysis of different models, algorithms, and strategies [8], [6]. Also, there is a strong need for high-performance simulation capabilities and efficient evaluation tools.
- *Production Requirements* comprise supports for easy deployment and configuration and scalability in large Grid communities [7]. Furthermore, in order to enable interoperation with other existing systems, compliance with open standards is needed.

Following, we depict the features and design of tGSF as a basis for reviewing requirements and best practices of real-world scheduling solutions. In this context, we highlight coverage and desiderata of current standards and protocols regarding various requirements and features in such implementations.

3.1 Features & Design

As depicted in Figure 1, the architecture of tGSF is divided into four layers, namely the Foundation Layer, the Commons Layer, the Site Layer, and the Grid Layer. Note that, although the usage of Foundation and Commons are mandatory and highly recommended, respectively, the different layers are loosely coupled and the inclusion of Site and Grid into a system instance is mutually optional.

3.1.1 Architecture. Here the different layers with their features and integration points into common standards are described.

Foundation Layer. The basis for tGSF is an event-driven *Kernel* responsible for time management and event dispatching. This kernel component uses an internal clock to isolate higher layers from the RTC and provide a common abstraction for time instants and periods. Furthermore, it provides an extensible type system which allows the handling of domain-tailored events.

The behaviour of tGSF however, does not depend on the kernel itself: in fact, it is determined by the *Runtime Environment*, which – depending

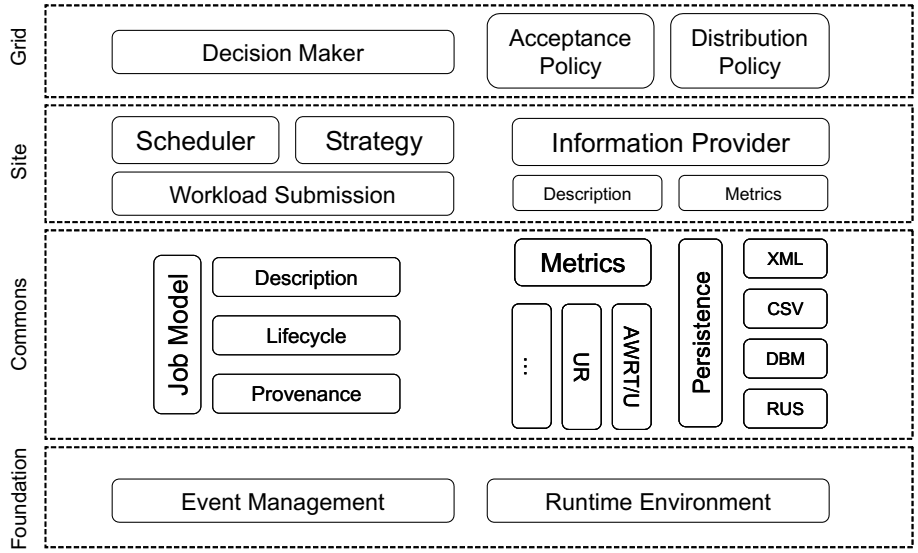

Figure 1. Architectural view of the Teikoku Scheduling Framework

on the implementation – allows the realisation of real-time as well as simulation or debugging systems.

Commons Layer. To provide general abstractions for management-related concepts, a *Job Model*, *Metrics*, and *Persistence* services are supplied in this layer.

The job model uses a holistic view on units of work within a Grid environment: it consists of a description of static job characteristics and resource requirements, a life-cycle which represents current and historic states of the job, and a provenance that denotes the track or route a job has taken from its original submission until final assignment and execution. In this context, JSDL [2] can be used to at least tackle the specification of static properties and resource constraints for a job.

The metrics services provide a unified interface to the measurement data during runtime: values regarding different aspects can be stored and retrieved, e.g. for supporting advanced decision strategies. Such metrics may include traditional performance objectives such as response time and utilisation as well as accounting information such as OGSA-UR [11].

The persistence services offer a common access to different storage mechanisms for recording and replaying metrics values, e.g. files and relational databases as well as standard resource usage services as defined by OGSA-RUS.

Site Layer. The management of a distinct set of resources is conducted by the Site Layer, which abstracts the implementation of scheduler functionality, see Figure 2. In addition to appropriate data structures for keeping information on the current schedule, it allows the application of assignment strategies during runtime, and offers a Service Provider Interface (SPI) towards traditional LRMSs. These interfaces might conform to the wide spread DRMAA [3] standard, but also to OGSA-BES [9], OGSA-HPCP [4], or more traditional protocols such as Globus WS-GRAM [5] or the POSIX batch management family of commands [10].

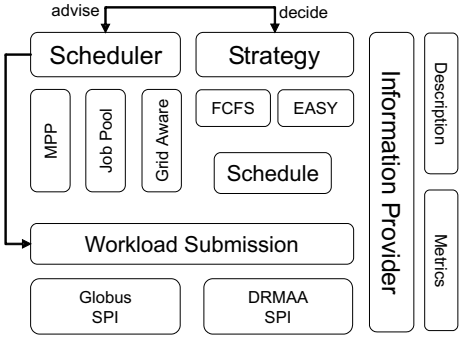

Figure 2. TGSF's Site Layer

Although the strategies calculate the assignment of work to resources, their results have advisory character: they propose their solutions to the scheduler which in turn decides on the actions to take. This allows the scheduler to consult multiple strategies at a time and select the most appropriate solution.

The information provider bundles static as well as dynamic site information such as number of resources, current utilisation etc. and exposes this data to the other Site Layer components and – at least partly – to the global environment.

Grid Layer. In order to enable an interaction component for the distribution of workload to other participating sites within a Grid environment, the Grid Layer provides facilities for the policy-based exchange of jobs.

Depending on the Grid environment's scheduler topology, the *Decision Maker* component has different responsibilities. In centralised or hierarchical environments, it's main task is the acceptance of jobs and their assignment to underlying resources or subordinate brokers. In decentralised scenarios, the redistribution to other decision makers is conducted additionally; the decision on the modalities is based on a set of

policies regarding acceptance and distribution of jobs. A basic standard to be used in this context is the WS-Agreement [1] specification.

3.1.2 Control Flow. In order to demonstrate the dynamics within TGSF during a job scheduling life-cycle, we give an overview on the control flow within and between the different components and layers. Note that we assume a topology of equitable, P2P-like arranged sites with autonomous operation characteristics.

A job enters the system through the Grid Layer's submission interface in the decision maker, either submitted by a local user or a decision maker from a remote site. The decision maker then decides—based on its policy set—whether to execute the job locally or to reject or redistribute it. For this purpose, it may use the local Information Provider to support its decision. In either case, it updates the job's life-cycle and provenance information.

In case of acceptance, the decision maker then hands over the job to the scheduler in the Site Layer. From this point on, the local scheduler is responsible for the job's execution. The scheduling process is supported by scheduling strategies that evaluate the current system state and consequences of resource assignments. Based on the strategies' advice, the scheduler is able to decide on the next actions to take, which can again result in redistribution of the job or a final assignment to selected resources.

While the scheduling strategies rely on the information which is provided by the metrics and the job model from the Commons Layer to generate an advice, the operation of the whole system is controlled by the runtime environment abstraction and the underlying event management from the Foundation Layer.

4. Bridging the Gap – The Grid Scheduling Architecture

The previous section gave some architectural details of the TGSF and pointed out several interfaces between services and layers where standardised protocols can be used to ensure a flexible and customisable application of the framework.

Although there are already a couple of possible solutions for workload submission, job description, metrics and persistence, for several important interfaces in the grid scheduling context either no or not adequate standards exist, yet. One goal of this section is to highlight these gaps and to define requirements for possible solutions (cf. Sub-section 4.1).

Furthermore, this section outlines the efforts necessary to bridge the gaps (cf. Sub-section 4.2) and links them with existing standards to provide a Grid scheduling framework to test the interoperability of Grid schedulers (cf. Sub-section 4.3). This interoperability test together with the theoretical work described in this paper provide the foundations to define the Grid scheduling architecture. A side-effect of our work is that we evaluate the relations of the different OGF specifications and groups in the scheduling domain, point out where things are missing, and help to sketch a broad picture.

4.1 Existing Gaps

4.1.1 Job Model. The description of a job is mainly covered by JSDL: many aspects such as resource requirements, execution parameters, and input/output data definitions are defined therein and its extension mechanism allows proprietary enhancements. However, JSDL does not support the description of user-specified and scheduling-related constraints such as requested start times, due dates and deadlines, or runtime estimations.

The tracking of a job's life-cycle is also currently not properly standardised. Although OGSA-BES defines a very basic state model, it does not relate to intermediate states during the process of scheduling a job. Furthermore, no common interface to access the entire life-cycle of a job exists; one can only retrieve its current state.

The same holds for storing and retrieving provenance data of a job. Especially in distributed architectures this information is essential not only to the user, but also highly beneficial for supporting the decision process of scheduling strategies.

4.1.2 Algorithm & Policy Model. Since the design of advanced scheduling strategies is a complicated task which is mostly conducted in research, an easy integration of developed results into available Grid scheduling products would be highly desirable.

To this end, a standardised interface for pluggable decision strategies is needed to enable the simple deployment of newly created algorithms into production systems. Besides that, such an approach would support an integrated development life-cycle for such algorithms, since both test/simulation environments as well as real-world systems would behave in a similar way. A possible first step towards such an interface could be the aforementioned advisory characterisation of strategies, where the scheduler requests possible solutions to reach an adequate decision.

The same problem applies for acceptance and distribution policies on the Grid interaction level.

4.2 Bridging the Gaps

From the architectural point of view, which is the focus of our work and hence this paper, solutions to overcome the issues related to the Job Model are those of interest since they prevent us from realising a Grid scheduling architecture with the characteristics given in the first section. The Algorithm & Policy Model related shortcomings outlined above need solutions, too, but those are currently not in the focus of our work as they are related to scheduler-internal architecture which we treat as a black box in the light of a general Grid scheduling architecture.

JSDL's extension mechanism has a negative and a positive effect. Negative is the fact that it can (and is) used to extend JSDL to realise proprietary enhancements which run counter to the idea of interoperability. Advantageous, however, is the extension mechanism to specify all the missing scheduling-related constraints.

To restrict the usage of JSDL to a limited number of attributes we decided to define a *JSDL Profile*, like it is e.g done by the OGSA-HPC group, to achieve an "a priori" agreement between interoperable schedulers. This profile defines in detail the supported JSDL attributes and how they are to be interpreted. Regarding the scheduling-related attributes which are not included in JSDL, we introduce the *Scheduling Description Language* (SDL). The goal is to provide a basic set of scheduling attributes that must be handled by Grid schedulers. Candidate attribute categories are time constraints, job dependencies, job priorities, scheduling objectives/preferences, data-dependent scheduling information, and queue-based scheduling information. These attributes may be referenced in SLAs negotiated between Grid schedulers, used for scheduling decisions, or exploited to execute a job by a local resource management system.

Regarding the life-cycle of a job, there is the OGSA-BES extension mechanism to its state model which allows the definition of domain-specific sub-states. But as far as the access to the life-cycle of a job and and its provenance data is concerned, only initial effort has been taken by the JSDL group.

4.3 Scheduler Interoperability

The interoperability test of Grid schedulers is another major step towards a Grid scheduling architecture. It makes use of Grid scheduling uses cases and requirements, the results from the TGSF evaluation, and integrates the JSDL Profile and the Scheduling Description Language with existing standards. The scenario requires that participating sched-

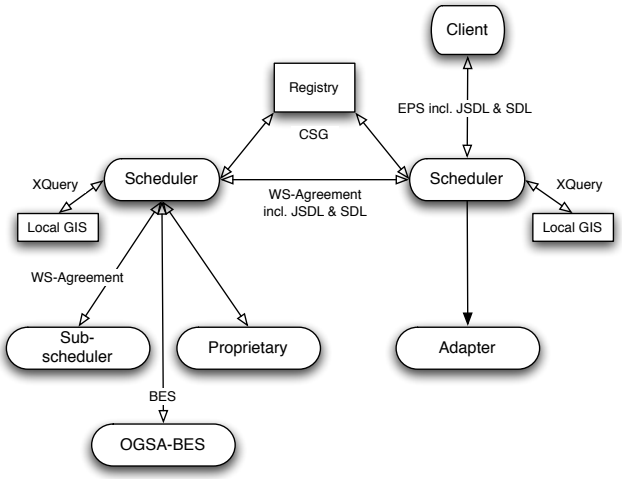

Figure 3. Interoperability of Grid Schedulers using OGF standards

ulers interoperate with each other in a case where one scheduler is unable to fulfil a scheduling request and delegates it to another scheduler.

The following interactions to reach agreement on the delegation of a scheduling decision are required (see Figure 3):

- Scheduler A, which realises an OGSA-RSS Execution Planning Service (EPS), cannot fulfil a scheduler request. The JSDL Profile-compliant job together with the SDL attributes is part of this request.
- The request is passed to Scheduler B, using WS-Agreement (please note that we assume that this request crosses administrative domains and we therefore rely on Service Level Agreements).
- Scheduler B checks its capabilities.
- Scheduler A and B agree/disagree on the conditions to fulfil the request (again using WS-Agreement).
- Scheduler B fulfils the scheduling request either directly by passing the job to an OGSA-BES service, by negotiating with another (local) scheduler, or by contacting some proprietary service.

Calls to a registry, which has OGSA-RSS' Candidate Set Generator capabilities, to check schedulers' capabilities and queries to local Grid Information Services (GIS) are also part of the information flow of the interoperability test.

Once the participating schedulers have implemented the interoperability interfaces and protocols and the test has been conducted, the GSA-RG will be able to finally define a Grid scheduling architecture in an standards-compliant landscape.

5. Status Quo & Outlook

The JSDL Profile and the SDL, which have been introduced in the previous section as potential instruments to fill the most important gaps en route to defining a standards-compliant, interoperable, and generic Grid scheduling architecture, exist as draft specifications. The GSA-RG is currently working on the finalisation of these documents together with other groups at the Open Grid Forum. This work is executed under the umbrella of an scheduler interoperability test where CoreGRID partners plan to implement the interoperability architecture sketched in Figure 3. In this test currently participate the Viola MetaScheduling Service[6] and the Grid Resource Management System[7], but other CoreGRID partners have already expressed their interest. The evaluation of this test together with the findings from the theoretical evaluations will finally result in the definition of a standards-compliant, interoperable, and generic Grid scheduling architecture.

Acknowledgments

This paper includes work carried out jointly within the CoreGRID Network of Excellence funded by the European Commission's IST programme under grant #004265.

[6] `http://www.viola-testbed.de`
[7] `http://www.gridge.org`

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. WS-Agreement - Web Services Agreement Specification. Grid Forum Document, GFD.108, Open Grid Forum, May, 2007.

[2] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification, Version 1.0. online [http://forge.gridforum.org/projects/jsdl-wg], November 2005.

[3] Roger Brobst, Waiman Chan, Fritz Ferstl, Jeff Gardiner, Andreas Haas, Bill Nitzberg, Hrabri Rajic, Daniel Templeton, John Tollefsrud, and Peter Trger. Distributed Resource Management Application API Specification 1.0. Recommendation GFD.22, Open Grid Forum, Lemont (IL), USA, April 2004.

[4] Blair Dillaway, Marty Humphrey, Chris Smith, Marvin Theimer, and Wasson Glenn. HPC Basic Profile, Version 1.0. online [http://forge.gridforum.org/projects/ogsa-hpcp-wg], August 2007.

[5] Ian Foster and Carl Kesselman. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprint for a Future Computing Infrastructure*, pages 259–278. Morgan Kaufman, San Mateo (CA), 1st edition, 1998.

[6] Ch. Grimme, J. Lepping, and A. Papaspyrou. Identifying Job Migration Characteristics in Decentralized Grid Scheduling Scenarios. In *Proceedings of the 19th International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Cambridge (MA), USA, November 2007. IASTED, ACTA Press. To appear.

[7] Christian Grimme, Tobias Langhammer, Alexander Papaspyrou, and Florian Schintke. Negotiation-based Choreography of Data-intensive Applications in the C3Grid Project. In *Proceedings of the German e-Science Conference (GES)*, Baden-Baden, Germany, May 2007. Max-Planck Society (online).

[8] Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. Prospects of Collaboration between Compute Providers by Means of Job Interchange. In E. Frachtenberg and U. Schwiegelshohn, editors, *Proceedings of the 13th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Lecture Notes on Computer Science (LNCS), Seattle (WA), USA, June 2007. Springer. To appear.

[9] A. Grimshaw, S. Newhouse, et al. OGSA Basic Execution Service Version 1.0. online [http://forge.gridforum.org/projects/ogsa-bes-wg], December 2006.

[10] IEEE Computer Society Portable Applications Standards Committee. Portable Operating System Interface (POSIX). Shell and utilities. In *Standard for Information Technology*, volume 6 of *The Open Group Base Specifications*. IEEE Press, 2004.

[11] R. Mach, R Lepra-Metz, and S. Jackson. Usage Record – Format Recommendation. online [http://forge.gridforum.org/projects/ur-wg], February 2007.

[12] N. Tonellotto, Ph. Wieder, and R. Yahyapour. A Proposal for a Generic Grid Scheduling Architecture. In S. Gorlatch and M. Danelutto, editors, *Proceedings of the Integrated Research in Grid Computing Workshop 2005*, CoreGRID Series, pages 227–239. Springer, 2007.

[13] R. Yahyapour and Ph. Wieder. Grid scheduling use cases. Grid Forum Document, GFD.64, Global Grid Forum, March, 2006.