On Scheduling in UNICORE - Extending the Web Services Agreement based Resource Management Framework

A. Streit, O. Wäldrich, P. Wieder, W. Ziegler

On Scheduling in UNICORE – Extending the Web Services Agreement based Resource Management Framework[*]

A. Streit[a], Oliver Wäldrich[b], Ph. Wieder[a], W. Ziegler[b]

[a]Central Institute for Applied Mathematics, Research Centre Jülich, 52425 Jülich, Germany

[b]Fraunhofer Institute SCAI, Department of Bioinformatics, 53754 Sankt Augustin, Germany

1. Introduction

Designing, building and using Grids generally implies that the resources involved are highly distributed, heterogeneous and managed by different organisations. These characteristics hamper the coordinated usage and management of resources within a Grid, which in turn motivates the development of Grid-specific resource management and scheduling solutions. Please refer for instance to [8] which collects a large variety of contributions to research and development for this specific topic.

The use case of the work we present here requires a Grid that has exactly the properties listed above: The VIOLA Grid [15] comprises distributed resources of different types which are owned by different organisations and have to be managed in a coordinated fashion. Although the resources have different owners the aspects of authentication and authorisation are of secondary importance to this work since the underlying middleware, UNICORE, provides all necessary security means required to realise a Virtual Organisation [3].

The focal point of this paper is the orchestration of resources. In our case various compute and network resources have to be co-allocated to form a virtual machine that enables the execution of distributed parallel applications. To achieve this goal, a meta-scheduler is needed which generates a schedule based on user requirements and guarantees that the requested resources are reserved in advance. Therefore we developed the VIOLA MetaScheduling Service, which is founded on the experience with the UNICORE WS-Agreement prototype [11], the MeSch meta-scheduler [10] and the UNICORE timer process [2, chapter 3.7.2 "Distributed Applications"]. The integration of UNICORE and the MetaScheduling Service provides a framework which fulfils the use case's requirements and lays the foundation for co-allocation of arbitrary resources.

Following the introduction (this section) we describe in Section 2 the resource management mechanisms of the UNICORE software and the resulting challenges with respect to the VIOLA use case. Based on this description we introduce in Section 3 the VIOLA MetaScheduling Service including its architecture and the internal processes. We conclude the paper with an outline of future work related to scheduling in UNICORE (see Section 4) and a short summary (see Section 5).

2. UNICORE Resource Management

The resource management in UNICORE is centred around two concepts: resource virtualisation and job abstraction [2]. These concepts are connected through the *UNICORE Resource Model*, which provides a unified mechanism to describe both resource requests and resource properties.

The virtualisation of resources manifests itself in UNICORE's *Virtual Site* (Vsite) that comprises
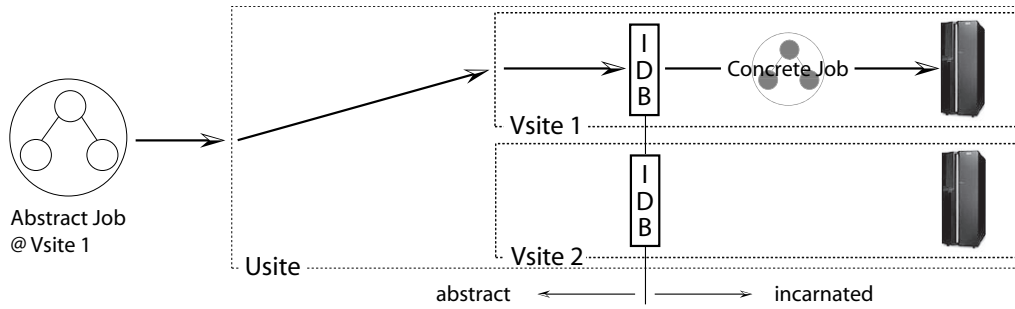
Figure 1. Two stages of a UNICORE job

a set of resources. These resources must have direct access to each other, a uniform user mapping, and they are generally under the same administrative control. A set of Vsites is represented by a *UNICORE Site* (Usite) that offers a single access point (a unique address and port) to the resources of usually one institution.

A UNICORE job passes two different stages during its lifetime (see Fig. 1), an abstract and an incarnated one. During the former the job description contains no site-specific information such as paths to executables or paths to data, and it is processable by every Usite, whereas during the latter the job representation includes all information necessary to actually execute the job on resource level. Between these two stages the abstract job representation is "translated"; a process that is called *incarnation* and is carried out at Vsite level. For incarnation as well as for the composition of jobs resource-specific information is needed. The information service that keeps and propagates this information is the *Incarnation Database* (IDB), a service deployed per Vsite. It maintains resource information according to the UNICORE Resource Model and provides inter alia information about *capability resources*, such as software distributions, and about *capacity resources*, such as processors or memory.

Most Grid scheduling actions within a UNICORE environment are currently carried out manually. Based on the classification in [12], ten actions in three phases are distinguished during the process of scheduling. Resource discovery and system selection are performed by the user. Entering the third phase called job execution (on Usite level) the abstract job representation has then its target Vsite assigned, as shown in Fig. 1. This implies that candidate resources have been pre-selected and mapped to the user's or application's resource requirements. In the current implementation these tasks are executed by the user via the UNICORE graphical client although the UNICORE concepts and models do not impose this modus operandi.

To satisfy the VIOLA meta-scheduling use case UNICORE needs to be at least extended to provide advance reservation capabilities. This can be realised by integrating WS-Agreement providers and consumers into UNICORE, as reported in [11], and has been implemented in the first phase of the MetaScheduling Service development as outlined in the following section.

3. Scheduling Arbitrary Resources

Scheduling resources with the objective of meeting a user's demand for a distinct Quality of Service (QoS), e.g. precisely timed availability of multiple resources, requires local scheduling systems with two indispensable capabilities in order to allocate resources efficiently:

- To publish the time-slots at which the resource is available.

- To make a reservation for a given start time and stop time.

At this stage there are just a few local scheduling systems with these capabilities, such as PBS Professional [9], CCS [7], EASY [13], or LSF. But we expect new systems appear (like for example the next version of LoadLeveler) as these capabilities are crucial for resource owners who intend to advertise their resources with guarantees for QoS to the Grid. Other approaches have been proposed to support co-allocation of resources based on simple queueing by the local scheduling systems. Following either a trial and error approach or extending the reservation times hoping to create a matching time-window for the job, these approaches deliver only best-effort results. The first approach keeps the local scheduling busy with requests and cancellation of requests, while the second one often results in wasting resources that are waiting for other resources to become available if the local queues are populated.

3.1. Interaction with Local Scheduling Systems

Local scheduling systems typically have different APIs with different behaviour. The handling of these differences is not part of the MetaScheduling Service, instead an Adapter Pattern [4] approach is used. This results in a specific light-weight adapter for each local scheduling system which is designed to facilitate the integration of new local scheduling systems. The adapters implement the (command line) interface to the local scheduling system on one end and the SOAP [14] interface to the MetaScheduling Service on the other end. The MetaScheduling Service itself is agnostic to the details of local scheduling systems and implements the SOAP interface which includes all functions necessary to carry out meta-scheduling:

- `ResourceAvailableAt` - returns the next possible start time for a job on a resource that matches the properties of a user's request.

- `Submit` - submits a job to a free slot identified before.

- `Cancel` - removes a component of a meta-job from a local scheduler.

- `State` - returns the state of a local reservation.

- `Bind` - submits IP addresses of allocated resources (at runtime).

- `Publish` - submits runtime information to the local system.

Only the first four functions are needed to generate a meta-schedule, the remaining two provide additional management functions. In case of network resources the `Bind` call allows to communicate the IP addresses of all the compute nodes which are allocated for a job once the job has been started. Thus the resource management system for the network resources is enabled to set up the necessary end-to-end network configuration with the required QoS. The `Publish` call finally is used to submit runtime information to the local systems, like information necessary to configure the MPI environment or wrappers for batch scripts to start-up a job.

The pseudo code in Listing 1 describes the algorithm to identify common free slots using the `ResourceAvailableAt` call. To speed up the process the `ResourceAvailableAt` calls are executed in parallel, hence the request for the next possible start time is performed in parallel for all adapters.

Listing 1: Pseudo code of the time-slot algorithm

```
set n = number of requested resources
set resources[1..n] = requested resources
set properties[1..n] = requested property per resource # number of nodes, bandwidth, time,...
set freeSlots[1..n] = null                             # start time of free slots
set endOfPreviewWindow = false
set nextStartupTime = currentTime+someMinutes          # the starting point when looking for free slots

while (endOfPreviewWindow = false) do {

  for 1..n do in parallel {
    freeSlots[i] =  ResourceAvailableAt(resources[i], properties[i], nextStartupTime)
  }

  for 1..n do {
    set needNext = false
    if (nextStartupTime != freeSlots[i]) then {
      if (freeSlots[i] != null) then {
        if (nextStartupTime < freeSlot[i]) then {
          set nextStartupTime = freeSlots[i]
          set needNext = true
        }
      } else {
        set endOfPreviewWindow = true
      }
    }
  }
}

if ((needNext = false) & (endOfPreviewWindow = false)) then return
  freeSlots[1] else return "no common slot found"
```

3.2. Capabilities of a MetaScheduling Service

There are several tasks a meta-scheduler should be able to perform in a resource management framework. It should for example be able to:

1. Allocate a single resource for a single application for a fixed period of time.

2. Co-allocate several resources for the same fixed period of time for single or multiple applications.

3. Allocate multiple resources for multiple applications for different fixed periods of time.

4. Allocate dedicated resources for either of the cases above.

The current version of the MetaScheduling Service is able to perform task 1. and 2. which allow to schedule a simple job to one resource in the Grid or to schedule a job composed of several components that need to be executed at the same time to multiple resources in the Grid. The third task, being essential to schedule workflows in the Grid without wasting resources, will be implemented in the next version of the service, just as the fourth task which is required to allocate resources with special capabilities, i.e. I/O nodes for parallel I/O.

3.3. Negotiation Process

In this section we describe the protocol the MetaScheduling Service uses to negotiate the allocation of resources with the local scheduling systems. The protocol is illustrated in Fig. 2. As Section 3.1 describes, the resource negotiation starts with the MetaScheduling Service acquiring a preview of the resources available in the near future from the individual scheduling subsystems. The consolidation of the previews from all systems results in a list of resources and the next possible start times for the job scheduled to these resources.

The algorithm terminates either if at least one resource no longer advertises a next possible start time within the look-ahead time-frame of the local scheduling system, or when a common time-slot is identified. As long as all resources return the next possible start times, the MetaScheduling Service calculates whether there is sufficient overlap between the returned slots to allocate all requested
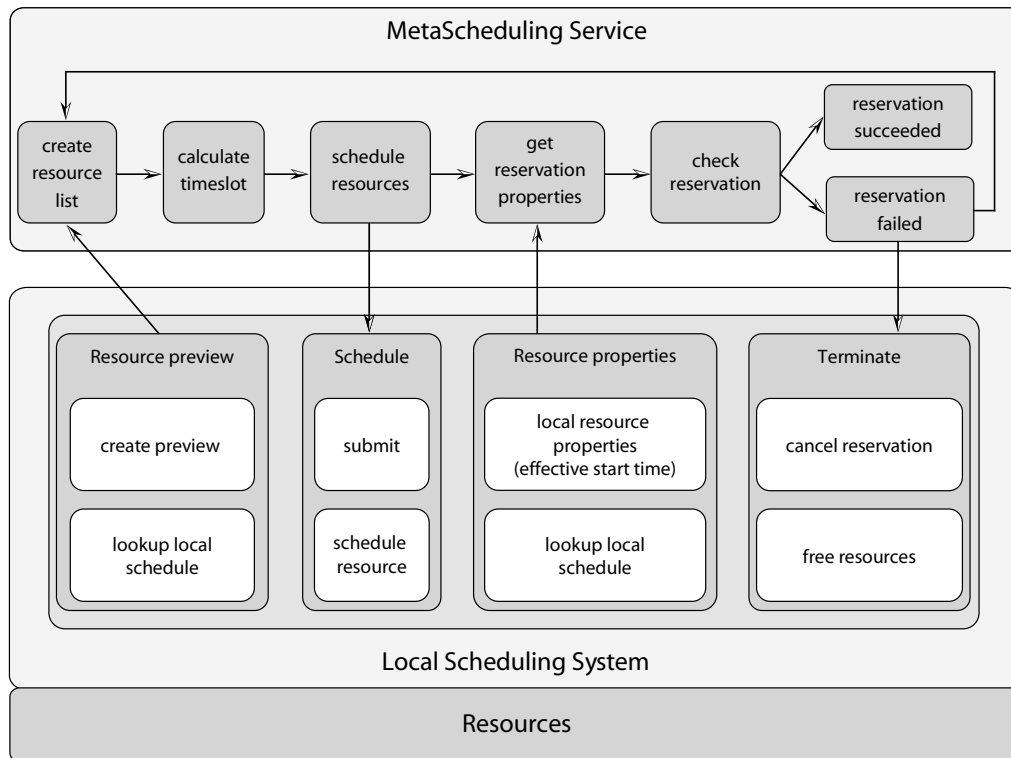
Figure 2. The negotiation process

resources. If not, the slot with the latest possible starting time is selected as starting point of the next preview query. If there is sufficient overlap, the next steps of the protocol are performed:

- The MetaScheduling Service submits the specific reservation requests with the calculated start time to the local scheduling systems.

- The adapters return a confirmation from the local scheduling systems, indicating the start time of the reservation and the properties (number of nodes, bandwidth, etc.).

- The MetaScheduling Service then checks whether the confirmations received match the requests.

Verifying the schedules is necessary because in the meantime new reservations may have been submitted to the local schedulers by other (local) users or processes, which may prevent the scheduling of the reservation at the requested time. If the MetaScheduling Service detects one or more reservations that are not scheduled at the requested time, all reservations will be cancelled. The latest effective start time of all reservations in the current scheduling cycle will be used as the earliest starting time in the next negotiation cycle. If this finishes successfully and all reservations are scheduled at the appropriate time on the local systems the co-allocation of the resources will be completed.

3.4. Integration into the UNICORE Environment

As shown in Fig. 3 the UNICORE system and the MetaScheduling Service are currently loosely coupled through the UNICORE client. A special plug-in for the UNICORE client allows the user

to specify his request for co-allocated resources to run a distributed job in the VIOLA testbed. The same plug-in also manages communication with the MetaScheduling Service. Once Unicore/GS, the service-oriented version of UNICORE, will be available, the integration will be rather at the level of the UNICORE Server. But even with the current level of integration it is possible to use UNICORE mechanisms to map the certificate a user presents to the client for authentication to the (probably) different uids of this user at the different sites of the testbed. This information is passed to the MetaScheduling Service along with a list of the requested resources and enables the service to do the reservations at the different sites with the correct local uid. The SOAP interface for communication between UNICORE client and MetaScheduling Service is using WS-Agreement [1] to exchange the details and conditions of the reservation:

- The client requests an agreement template from the MetaScheduling Service.

- The MetaScheduling Service replies with the template containing information about the available resources (UNICORE Vsites).

- The service sends an agreement proposal containing the resource request information.

- If the (co-)allocation is successful, the MetaScheduling Service returns the final agreement.

Once the (co-)allocation has been completed with the reception of the final agreement the UNICORE client creates the abstract UNICORE job that contains a request specified by the user, submits the job to the UNICORE gateway and invokes the normal UNICORE processing of a job as described in Section 2.

In the current framework there is neither a resource detection service nor a resource selection service, thus the user has to explicitly specify the resources he wants to use for a job, i.e. sites and corresponding properties of the reservations. In a later version a broker function will be integrated, allowing a user to describe his request by specifying resource properties and leaving the selection of appropriate resources to the broker. This also relieves the user from resource discovery and system selection if he does not want to control these tasks. The MetaScheduling service then may use the selection suggested by the broker to negotiate the co-allocation or other QoS requirements specified by the user.

4. Future Work

The performance of the the MetaScheduling Service – integrated into UNICORE middleware – will be explored in a performance evaluation. For this purpose the distributed test environment within the VIOLA project is used as a basis. Within this test environment several compute resources are integrated, each one equipped with a different resource management system capable of advance reservation. We will use EASY, PBS Professional, and CCS, while the latest version of LoadLeveler Multi-Cluster (LL-MC), which also supports advance reservations, will be integrated as soon as the production version becomes available.

This test environment will be used to evaluate the performance of the MetaScheduling Service architecture with different baseline scenarios, applied workloads, and meta-job requests. Besides measuring the costs of scheduling for placing meta-jobs with advance reservations, additional criteria will be evaluated. For example, the success of the MetaScheduling Service in placing meta-jobs can be measured by the rejection rate for such jobs with given start times, and by the average waiting time for meta-jobs, which should be placed by the MetaScheduling Service as soon as possible after
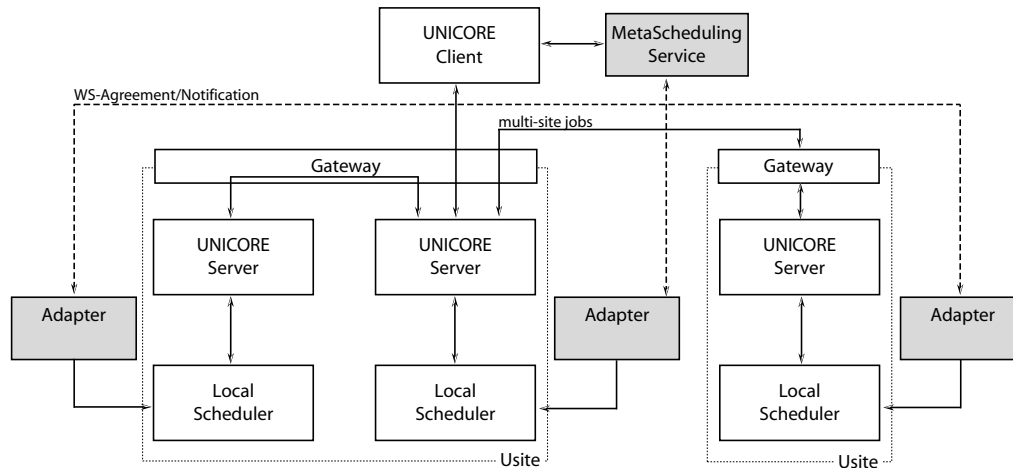
Figure 3. The meta-scheduler architecture

their submission. Additional metrics will be used if meta-jobs are described only by their total size of requested processors and not with a by partitioning. The local schedulers' performance is also influenced by the MetaScheduling Service, as the sub-parts of meta-jobs are placed as advance reservations and thereby influence the local scheduler when scheduling local batch jobs. Hence, the slowdown or average response time of local jobs has to be measured, too.

In addition we will enhance the MetaScheduling Service with Quality of Service and enhanced Service Level Agreements (SLA) functions. In particular negotiating and guaranteeing resource usage will be a key element to bridge the gaps of existing Grid systems as e.g. identified in the NGG2 report [6]. One approach to include such functionality is to apply the planning-based scheduling approach as described in [5]. Queueing-based scheduling considers only the present use of resources. In contrast, planning-based scheduling plans the present and future resource usage by assigning proposed start times to all submitted jobs and updating these whenever the schedule changes. This concept makes it easy to support guaranteed resource usage through advance reservation. The planning-based scheduling approach is implemented in the resource management software CCS [7] and has proved its abilities and benefits in recent years. CCS also provided the basis for an initial proof-of-concept implementation of a meta-scheduler within the UNICORE Plus project [2, chapter 3.7.2 "Distributed Applications"].

5. Conclusion

Coordinated and guaranteed resource provision with QoS and SLA support is a key element of modern resource management and scheduling solutions for Grid environments. UNICORE and its WS-Agreement based resource management framework already provide many desired capabilities. Within the VIOLA project a MetaScheduling Service was developed, which uses advance reservations on local systems for placing multi-site jobs onto the Grid.

In this paper we presented the integration of the MetaScheduling Service into the UNICORE Grid system, work that is carried out to provide meta-scheduling functions for the VIOLA testbed. Details of the implementation and the meta-scheduling processes were given. An even tighter integration with UNICORE will become possible once the Unicore/GS software will be available.

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification, July 2005. Online: https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/16.

[2] D. Erwin, editor. *UNICORE Plus Final Report – Uniform Interface to Computing Resources*. UNICORE Forum e.V., 2003. ISBN 3-00-011592-7.

[3] I. Foster, C. Kesselmann, J. M. Nick, and S. Tuecke. The Pysiology of the Grid. In F. Berman, G. C. Fox, and A. J. G. Hey, editors, *Grid Computing – Making the Global Infrastructure a Reality*, pages 217–249. John Wiley & Sons Ltd, 2003.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC Resource Management Systems: Queuing vs. Planning. In D. G. Feitelson and L. Rudolph, editors, *Proc. of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2003.

[6] K. Jeffery et al. Next Generation Grids 2 – Requirements and Options for European Grids Research 2005-2010 and Beyond, 2004. Online: `ftp://ftp.cordis.lu/pub/ist/docs/ngg2_eg_final.pdf`.

[7] A. Keller and A. Reinefeld. Anatomy of a Resource Management System for HPC Clusters. In *Annual Review of Scalable Computing*. Singapore University Press, 2001.

[8] J. Nabrzyski, J. Schopf, and J. Weglarz, editors. *Grid Resource Management – State of the Art and Future Trends*. Kluwer Academic Publishers, 2004.

[9] PBS Professional. Website. Online: `http://www.altair.com/software/pbspro.htm`.

[10] G. Quecke and W. Ziegler. MeSch – An Approach to Resource Management in a Distributed Environment. In *Proc. of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000)*, volume 1971 of *Lecture Notes in Computer Science*, pages 47–54. Springer, 2000.

[11] M. Riedel, V. Sander, Ph. Wieder, and J. Shan. Web Services Agreement based Resource Negotiation in UNICORE. In *Proc. of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, pages 31–37. CSREA Press, June 2005.

[12] J. Schopf. Ten Actions When Grid Scheduling – The User as a Grid Scheduler. In J. Nabrzyski, J. Schopf, and J. Weglarz, editors, *Grid Resource Management – State of the Art and Future Trends*, pages 15–23. Kluwer Academic Publishers, 2004.

[13] J. Skovira, W. Chan, H. Zhou, and D. Lifka. The EASY — LoadLeveler API Project. In D. G. Feitelson and L. Rudolph, editors, *Proc. of 2nd Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer, 1996.

[14] Simple Object Access Protocol Specification, SOAP Specification version 1.2. Website. Online: `http://www.w3.org/TR/soap12`.

[15] VIOLA - Vertically Integrated Optical Testbed for Large Applications in DFN, 2005. Web site. Online: `http://www.viola-testbed.net`.