

GWD-I  
Category: Informational  
Grid Scheduling Architecture Research Group (GSA-RG)  
draft-ggf-gsa-usecase-1.0

**Editors:**  
Ramin Yahyapour,  
Philipp Wieder,  
Andrea Pugliese,  
Domenico Talia,  
Jaegyeon Hahm  
July 19, 2004

# Grid Scheduling Use Cases

## Status of this Memo

This document provides information to the community regarding the Grid scheduling use case scenarios used in the definition of a Grid Scheduling Architecture (GSA-RG). Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

## Abstract

Grids will provide a large variety of complex services. The interactions of those services require an extensible and integrated resource management. Although such a coordinated scheduling of services is currently not readily available. Access to resources is typically subject to individual access, accounting, priority, and security policies of the resource owners. Those policies are typically enforced by local management systems. Therefore, an architecture that supports the interaction of independent local management systems with higher-level scheduling services is an important component for Grids. Further, user of a Grid may also establish individual scheduling objectives. Future Grid scheduling and resource management systems must consider those constraints in the scheduling process. Taking into account different policies is also important for the implementation of various economic and business models.

The goal of the Grid Scheduling Architecture research group (GSA-RG) is to define a scheduling architecture that supports cooperation between different scheduling instances for arbitrary Grid resources. Considered resources include network, software, data, storage and processing units. The research group will particularly address the interaction between resource management and data management. Co-allocation and the reservation of resources are key aspects of the new scheduling architecture, which will also include the integration of user or provider defined scheduling policies.

The group will begin with identifying a set of relevant use-cases based on experiences obtained by existing Grid projects. Then, it will determine the required components of a modular scheduling architecture and their interactions.



**GLOBAL GRID FORUM**  
**office@gridforum.org**  
**www.ggf.org**

## **Full Copyright Notice**

Copyright © Global Grid Forum (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## **Intellectual Property Statement**

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director (see contact information at GGF website).

## Contents

1	Introduction.....	4
2	Scheduling complex workflows.....	5
2.1	Summary .....	5
2.2	Customers .....	5
2.3	Scenarios.....	5
2.4	Involved resources .....	6
2.5	Functional requirements.....	6
2.6	Workflow of Scheduling Process.....	7
2.7	Involved Scheduling Components/Services .....	8
2.8	Failure Considerations .....	9
2.9	Security Considerations .....	9
2.10	Accounting Considerations.....	9
2.11	Performance Considerations .....	10
2.12	Use case Situation Analysis.....	10
2.13	References.....	11
3	Application-Oriented Scheduling in the KNOWLEDGE GRID .....	12
3.1	Summary .....	12
3.2	Customers .....	13
3.3	Scenarios.....	13
3.4	Involved resources .....	14
3.5	Functional requirements.....	14
3.6	Workflow of Scheduling Process.....	14
3.7	Involved Scheduling Components/Services .....	15
3.8	Failure Considerations .....	15
3.9	Security Considerations .....	15
3.10	Accounting Considerations.....	15
3.11	Performance Considerations .....	15
3.12	Use case Situation Analysis.....	15
3.13	References.....	15
4	GRASP (Grid Resource Allocation Services Package).....	18
4.1	Summary .....	18
4.2	Customers .....	19
4.3	Scenarios.....	19
4.4	Involved resources .....	20
4.5	Functional requirements.....	20
4.6	Workflow of Scheduling Process.....	21
4.7	Involved Scheduling Components/Services .....	21
4.8	Failure Considerations .....	22
4.9	Security Considerations .....	22
4.10	Accounting Considerations.....	22
4.11	Performance Considerations .....	22
4.12	Use case Situation Analysis.....	23
4.13	References.....	23
5	Editor Information .....	24

## 1 Introduction

One of the first milestones of the GSA-RG's charter is the identification of relevant use-cases for Grid scheduling.

This document is a collection of the use case scenarios contributed by GSA-RG participants or solicited from others.

Based on this document the GSA-RG will identify and specify common requirements to support the creation of Grid schedulers which serve the use-cases. This information will be used to identify components, services and protocols for a Grid scheduling architecture. Services and protocols from other GGF groups are considered as potential basic building blocks of such an architecture and will be used wherever possible.

Note, that it is not the task of the Research Group to define protocols or algorithms. Instead, the RG identifies the requirements for Grid scheduling, designs a suitable Grid scheduling architecture including existing services as well as currently missing components and their interaction.

## **2 Scheduling complex workflows**

### **2.1 Summary**

Many Grid applications require the coordinated processing of complex workflows which includes scheduling of heterogeneous resources within different administrative domains. Here, a typical scenario is the coordinated scheduling of computational resources in conjunction with data, storage, network and other available grid resources, like software licenses, experimental devices etc. The Grid scheduler should be able to coordinate and plan the workflow execution. That is, it should reserve the required resources and create a complete schedule for the whole workflow in advance.

In addition, cost management and accounting have to be considered in the scheduling process.

### **2.2 Customers**

This use case is of interest for a wide variety of costumers namely every Grid user who wants to process complex workflows. For instance, the presented use-case is common in climate-research, and high-energy physics.

### **2.3 Scenarios**

Since this use case defines the general requirements to schedule complex workflows a broad variety of scenarios is possible. This includes the “classical” example of scheduling a computational job including network, data, software and storage and also covers examples like Grid based steering of simulations or experiments.

A typical example would be the following user request:

- A specified architecture with 48 processing nodes,
- 1 GB of available memory, and
- a specified licensed software package are required
- for 1 hour between 8am and 6pm of the following day.
- In addition, a specific visualization device should be available during program execution, which requires
- minimum bandwidth between the visualization device and the main computer during program execution
- The program relies on a specified data set from a data repository for input.
- The user wants to spend at most 5 Euro and
- prefers a cheaper job execution over an earlier execution.

A Grid scheduler should be able to generate a complete schedule for the execution of this job including all resources required for implicit actions before and after the actual job start for data management. However, this example should be considered as a quite simple scenario. In a real application it could easily be extended to contain additional workflow steps. The Grid scheduler should take the allocation on all required resource types into account and if requested should create advance reservations. Figure 2-1 shows an example of the anticipated scheduling output of a Grid scheduler.

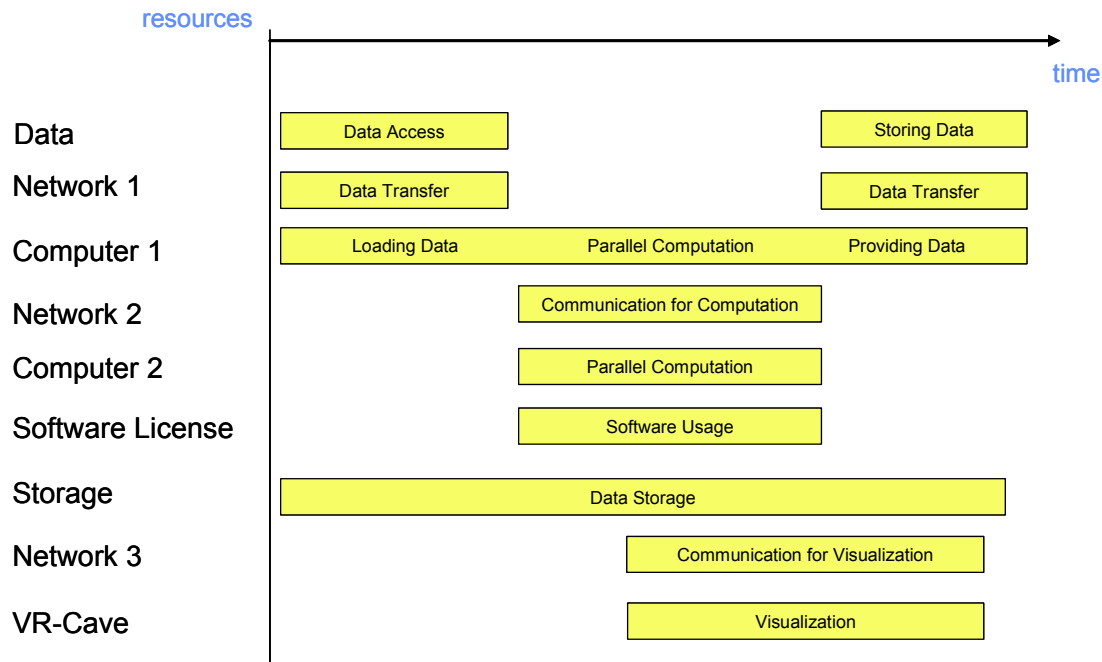


Figure 2-1: Example schedule

## 2.4 Involved resources

All kinds of available resources may be requested by the user, as long as the necessary means are in place to integrate them into the scheduling process. Figure 2-1 shows the usage of resources such as computing, data, storage, network and software resources, as well as special devices. But it can also be anticipated that services, sensors or even humans may be treated as resources in a Grid scheduling context.

## 2.5 Functional requirements

1. **Authentication, authorization, user right delegation & job integrity verification.** Authentication and authorization are essential for every Grid based job submission scenario. To enable the scheduler to act on behalf of the user the respective rights have to be delegated from the user to the scheduler. This use case also requires that the integrity of a job (parts of the job) can be verified anytime during the scheduling process.
2. **Job parsing & validation.** The job description has to be parsed and formally validated (job pre-processing).
3. **Information retrieval (static & dynamic).** To map the resource requests contained in the job description onto available resources, information about the resources and their status has to be retrieved from appropriate entities (and offered by these entities). It should be possible to gather static ("static" with respect to the runtime of the job) and dynamic resource information separately to restrict the time-consuming dynamic information retrieval.

4. **Resource pre-selection.** To avoid information queries on resources which do not fulfil policy constraints defined by the user or which are definitely not capable of fulfilling a resource request (why should one ask for information about the current system state if the system has less processors than required by the user) a set of resources should be selected based on those so-called “static” resource information.
5. **Service choreography, management.** It might be useful to have mechanisms which allow to choreograph/manage the services representing the pre-selected resources on different levels to obtain the desired dynamic information faster and more reliable (see 2.6 for the chronology of the scheduling process).
6. **Scheduling.** A schedule has to be generated based on the information about the job and the resources, accounts, etc.
7. **Advance reservation/agreement negotiation.** It is essential to meet time or precedence requirements defined by the workflow. Therefore one has to reserve in advance the resources selected by the schedule to guarantee the proper execution of the workflow. One approach to achieve this is specified by the GRAAP-WG [1], called Web Service Agreement. This specification defines a language/protocol to negotiate agreements between service provider and consumer.
8. **Workflow execution/processing.** The job has to be processed. It is assumed that the local resource managers execute the atomic entities a job is made of, but to process the workflow or parts of it, a workflow engine or processor is needed.
9. **Billing/accounting**
10. **Failure management.** This is essential not only to have an instrument to monitor and possibly reschedule jobs in case of failure within the system, but also to provide users with information and tools to manage such failure situations.

## 2.6 Workflow of Scheduling Process

The different steps of the scheduling process are described in this section referring to the example introduced in Section 2.3 (For each step the services needed are listed in brackets, see Section 2.7):

1. **Composition and submission the job request.** The job description is generated and transferred to an entity capable of processing its contents. In case of the example a job will be generated that contains the resource requests and constraints listed in Section 2.3. With respect to this use case no specific language to describe the job request is demanded. (Services 1 and 2)
2. **Pre-processing of the job request.** The job request has to be parsed and validated if possible. If the entity pre-processing the job is unable to do so it may try to translate the job to a suitable description. (Services 2 and 5)
3. **Gathering of static resource information.** Some service is needed which gathers static information about the resources<sup>1</sup>. This service may be an information

---

<sup>1</sup> Information is called static if it is known to be valid after the job has terminated. This may be the case e.g. concerning certain software available on a system, the maximum number of CPUs of a compute cluster, etc.

- service or a database. It is also possible that some Web Service Resource Property [4] is queried to gather static information about the service. Concerning the example it is assumed that this processing step identifies a pool of 800 resources of all requested kinds. (Services 3, 4, 8 and 9)
4. **Pre-selection of resources.** Based on the information collected in Step 3 algorithms are used to limit the number of resources which are potentially capable of participating in the workflow's processing. With regard to the example this may cut down resource candidates to 30 since e.g. some systems may not have 48 processors, may not offer the software requested or the respective system is maintained the next day. (Services 3)
  5. **Query of dynamic resource information.** The dynamic query delivers information like whether the current load of the machine allows to allocate 48 processors (this is different from Step 4, where resources are sorted out because they consist of less than 48 processors). This again limits the number of potential resources which are actually used in the next step to process the schedule. (Services 3, 4, 8 and 9)
  6. **Generation of schedule and initialization of required reservations.** Based on the resource information gathered in the previous steps a schedule is generated (e.g. as shown in Figure 2-1). It is then attempted to reserve the necessary resources in advance, a process which may fail several times due to the complexity of the workflow and the number of dependencies between the reservations needed. A failed negotiation with the resources chosen may lead to re-scheduling possibly with a preceding step 5. (Services 2 and 6)
  7. **Execution of workflow.** Once the schedule as shown in Figure 2-1 is confirmed it is processed and executed. In case of the example at first data is taken from some storage system and transferred via network 1 to computer 1. If no error occurs the workflow is executed until the last chunk of resulting data is written via network 1 to storage. (Services 2 and 7)
  8. **Completion of workflow.** This includes the finalization of accounting and billing as well as the delivery of the data the job produced. (Services 1, 2, 8 and 9)

## 2.7 Involved Scheduling Components/Services

The following services are required (Please note that this does not imply a separate service implementation for every entity listed here. The term service is used in the sense of some functionality provided by a certain software component, which may integrate several services. For each service the scheduling process steps it is involved are listed in brackets, see Section 2.6):

1. User or an agent acting on-behalf of a user (Scheduling process steps 1. and 8. The user/agent may also be involved in adjustments of the workflow if the systems permit that. This may happen at different steps, e.g. due to some failure condition)
2. Scheduling and resource management service (Scheduling process steps 1., 2., 3., 6., 7. and 8.)
3. Brokering service (Scheduling process steps 3., 4. and 5.)
4. Information service (Scheduling process steps 3. and 5.)
5. Translation service (Scheduling process step 2.)



6. Negotiation service (Scheduling process step 6.)
7. Execution service (Scheduling process step 7.)
8. Accounting service (Scheduling process steps 3., 5. and 8.)
9. Billing service (Scheduling process steps 3., 5. and 8.)

## 2.8 Failure Considerations

Based on 2.6 the following failures have to be taken into consideration:

- (*Processing of the job request*)
  1. The parser does not support the format of the job
  2. The job request is not valid.
- (*Gathering of static resource information*)
  1. The information source(s) needed to gather static information are not available.
- (*Pre-selection of resources*)
  1. Pre-selection of resource prevents workflow from being executed since resource requests already cannot be fulfilled.
- (*Query dynamic resource information*)
  1. The information source(s) needed to query dynamic information are not available.
- (*Generation of schedule & initialization of required reservations*)
  1. Requested resources are not available. The result of the dynamic resource query indicates that one or many of the resources requested are not available (maybe due to local resource manager failures, ..)
  2. Precedence relations/time constraints cannot be met. The initialization of reservations required by the schedule fails for one/many resources.
  3. Time out. No schedule could be generated within a pre-defined timeframe.
- (*Execution of workflow*)
  1. The execution of the workflow may fail for different reasons like e.g. temporary system unavailability, unrecoverable errors in the user code, etc.

Failures like unavailability of services, network, etc. are not considered here since those are use case independent failures.

## 2.9 Security Considerations

The functional requirements list the four most prominent security features demanded by this use case (see Section 2.5, bullet 1.). In general it has to be noted that protection of the user's identity, the job's integrity and the confidentiality of information has to be warranted throughout the whole process described here.

## 2.10 Accounting Considerations

- **Local domain accounting.** The use case described here does not define any demands concerning additional accounting mechanisms in addition to what is already implemented locally. But accounting information provided by the local resource administrators may have implications on the scheduling decisions so that e.g. specific resources are not available due to temporary local restrictions. To consider these information in the scheduling process they have to be available

- through the extended information service/broker (which implies an appropriate interface).
- **Inter-domain accounting.** The accounting/billing service is in the light of this use case a black box providing interfaces to send/receive accounting/billing information. Of greater interest are the information itself and the resulting brokering/scheduling decisions as well as the integration of an accounting/billing system into the system derived from this use case. It is suggested to refer to other activities at GGF (like GESA [2] and SA3 [3]) and work carried out in projects.

### **2.11 Performance Considerations**

The main impact on the performance of the whole process as described in Section 2.6 has the communication between the involved components/services. This includes the following items:

- **Scalability.** If the amount of resources which are part of a Grid increases, the communication between local resource managers and the scheduling service or the extended information service may have a negative impact on the overall system performance. Solutions like information caching (e.g. based on WS-Notification [4]) may be applied.
- **Choice of the service programming model.** Assuming that instances of that use case are performed in a Web Service based environment using SOAP [5] to exchange messages, one has to be aware that the performance is in general seen to be worse than that of other solutions like e.g. CORBA [6].
- **Communication failure.** In a service-oriented architecture as described above the failure of communication between services is not unusual. To realize a reliable system and enforce a certain level of service quality (and therefore increase performance), mechanisms are needed to manage services. One activity which is to be monitored here is the Web Services Distributed Management TC [7].

The performance impact of the resource request – resource offer mapping and the schedule generation is highly influenced by the performance of the implemented algorithms, but also by the estimated number of involved resources.

### **2.12 Use case Situation Analysis**

Diverse research and development activities are underway to find solutions for scheduling complex workflows as described in this use case, but no consistent and broadly applicable solution is available yet. It is envisaged that the Grid Scheduling Architecture Research Group will define an architecture which, once implemented, will provide the functions required by this use case.

It is of particular interest that the scheduling architecture derived from this (and other) uses case(s) is as much independent from the resources involved as possible.

### **2.13 References**

- [1] Grid Resource Allocation Agreement Protocol,  
<https://forge.gridforum.org/projects/graap-wg/>.
- [2] Grid Economic Services Architecture Working Group,  
<https://forge.gridforum.org/projects/gesa-wg/>.
- [3] Site Authentication, Authorization, and Accounting Requirements Research Group,  
<https://forge.gridforum.org/projects/saaa-rg/>.
- [4] OASIS Web Services Notification TC, <http://www.oasis-open.org/committees/wsn/charter.php>.
- [5] W3C XML Protocol Working Group, <http://www.w3.org/2000/xp/Group/>.
- [6] Gokhale, Kumar, and Sahuguet, “Reinventing the Wheel? CORBA vs. Web Services”, <http://www2002.org/CDROM/alternate/395/>.
- [7] OASIS Web Services Distributed Management TC, <http://www.oasis-open.org/committees/wsdm/charter.php>.

### 3 Application-Oriented Scheduling in the KNOWLEDGE GRID

#### 3.1 Summary

The KNOWLEDGE GRID (K-GRID) is an architecture built atop basic Grid middleware services that defines more specific services for the definition, composition, validation and execution of knowledge discovery applications over Grids, and for storing and managing discovered knowledge [1, 2]. The K-GRID *Resource Allocation and Execution Management Service (RAEMS)* is a service used by the KNOWLEDGE GRID to map applications onto available resources and to coordinate their execution. The K-GRID scheduler is part of the RAEMS; it can be seen as an “application agent” associated to each application to be executed. Indeed, the scheduler produces job assignments (along with timing constraints) for each application, with the goal of improving its performances, on the basis of knowledge or prediction about computational and I/O costs. Afterwards, it follows each application execution to adapt generated schedules to new information about job status and available resources. Moreover, since in realistic Grid applications it is generally infeasible to specify all the details of applications at composition time, the KNOWLEDGE GRID scheduler allows the definition and use of *abstract hosts*, i.e. hosts whose characteristics are only partially known, and that can be matched to different concrete ones [3].

Therefore, the main objectives of the scheduler are:

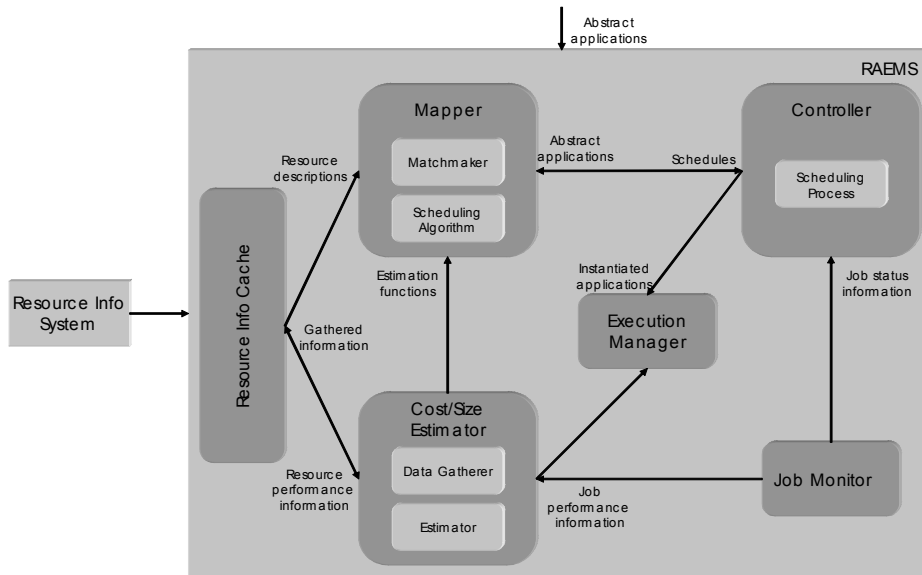
- *Abstraction from computational and network resources in application composition.* With the use of abstract hosts, users are allowed to disregard low-level execution-related aspects, and to concentrate more on the structure of their applications.
- *Application performance improvement.* Given a set of available hosts, schedules are generated trying to minimize applications’ completion times.

Besides the scheduler, the K-Grid’s RAEMS includes an Execution Manager, used to translate the output of the scheduling process into submissions to basic Grid services, and a Job Monitor that follows the execution of submitted jobs and notifies the scheduler about significant events occurred. Each K-Grid node has its own scheduler which is responsible for instantiating a new application agent for each scheduling request coming from the same or different nodes.

The architecture of the scheduler [4] comprises three main components (Figure 3-1):

- *Mapper.* It computes schedules employing a scheduling algorithm and making use of resource descriptions and computational and I/O cost evaluations.
- *Cost/Size Estimator.* It builds the I/O and computational cost estimation functions. The CSE comprises *data gathering* modules collecting dynamic information about current and future availability and performance of resources, and *estimation modules* dealing with the actual construction of estimation functions, on the basis of the perceived status of resources w.r.t. time.
- *Controller.* It guides the scheduling activity by receiving abstract applications, requesting the corresponding schedules to the Mapper, and ordering the execution of scheduled jobs to the Execution Manager. The Controller also receives

notifications about significant events occurred and re-schedules unexecuted parts of the application.



**Figure 3-1: K-GRID scheduler architecture**

The scheduler modules are extensible as they provide an open interface allowing to *plug-in* user-defined functionalities and behaviours. The scheduler can load modules implementing scheduling algorithms and matchmaking functionalities (in the Mapper), scheduling processes (in the Controller), and data gathering and cost estimation activities (in the Cost/Size Estimator). Each module can refer to its own description of resources. This makes the scheduler potentially useful in Grid frameworks different from the KNOWLEDGE GRID. For instance, cooperation among different schedulers could be implemented in the scheduling process, and resource and applications' descriptions could be properly designed to include the needed information.

### 3.2 Customers

The target customers of the KNOWLEDGE GRID scheduler are mostly Grid users who want to perform knowledge discovery processes on Grids. However, since the scheduler is not tightly coupled with the KNOWLEDGE GRID architecture, its use can be seamlessly extended to other application domains.

### 3.3 Scenarios

- *Application submission.* The scheduler interprets user's request and finds a suitable schedule for it by matching resource requirements with concrete resource descriptions, and trying to minimize the application completion time.
- *Restart on failure.* Both computation and communication jobs are automatically observed during the execution, and a re-scheduling policy can be implemented in the scheduler Controller.

- *Extension.* The scheduler can load modules implementing different functionalities, each of which can be based on a different way of characterizing resources.

### 3.4 Involved resources

The extensibility of the KNOWLEDGE GRID scheduler allows the use of virtually any kind of resource needed by the users; the only limitations are those of resource providers.

### 3.5 Functional requirements

- *Information retrieval.* The scheduler must be able to connect to external resource information services to retrieve data about (current and future) availability and performance of resources.
- *Application parsing and validation.* The scheduler must parse and validate the scheduling requests w.r.t. their structure and w.r.t. the actual possibility to instantiate them.
- *Resource pre-selection.* The available resources must be preliminarily filtered to retain only those actually usable for the application.
- *Scheduling.* The scheduler must support a *scheduling process*, i.e., the sequence of actions to be taken in coincidence with particular events, and a *scheduling algorithm*, defining the way in which jobs are assigned to resources.
- *Failure management.* The scheduling process must be *dynamic with re-scheduling*, i.e., the scheduler is invoked initially and then, during application executions, it is invoked again as a consequence of significant events occurred, to re-schedule unexecuted parts of the application.
- *Extensibility.* It must be possible to extend the scheduler functionalities with personalized ones based on different application scenarios and Grid structures.

### 3.6 Workflow of Scheduling Process

For each application to be scheduled, the scheduler instantiates a different Controller. Moreover, the following logical steps are performed:

1. The Cost/Size Estimator gathers data about characteristics and performances of available resources and builds the cost estimation functions (this step can be done offline).
2. The Matchmaker selects resources usable to execute the jobs composing the application, using information coming from the Resource Information Cache.
3. The Mapper evaluates a certain set of possible schedules, using information coming from the Estimation modules of the Cost/Size Estimator, and chooses of the one minimizing the completion time.
4. The Controller requests job execution to the Execution Manager.
5. The Controller waits for job status notifications from the Job Monitor or new information about availability and performance of resources and adapts the schedule to such changes.

### **3.7 Involved Scheduling Components/Services**

Based on the general Grid scheduling architecture as defined in October, 2002 GSA document, the following services are involved in the activity of the KNOWLEDGE GRID scheduler:

- *Data and Network Management* services;
- *Job Supervisor* service;
- *Information* service (static and forecasted).

### **3.8 Failure Considerations**

The KNOWLEDGE GRID scheduler handles job failures as described in Section 2.3.

### **3.9 Security Considerations**

Security in the KNOWLEDGE GRID scheduler is demanded to other KNOWLEDGE GRID services; it is essentially based on GSI.

### **3.10 Accounting Considerations**

Accounting in the KNOWLEDGE GRID scheduler is demanded to other KNOWLEDGE GRID services.

### **3.11 Performance Considerations**

The KNOWLEDGE GRID scheduler caches resource information and strongly indexes them in order to obtain the data access performance needed during the scheduling activity. In addition, due to the inherent intractability of the scheduling problem to be dealt with, one of the most important requirements of the scheduling heuristics is a suitable effectiveness/efficiency trade-off.

### **3.12 Use case Situation Analysis**

We have designed a complete scheduling model and implemented the architecture described in Section 2.1 for its support. The scheduler is currently within the context of the KNOWLEDGE GRID, but its structure and openness prove suitable for more general scheduling scenarios. The study of suitable scheduling heuristics for different kinds of applications and Grids is currently underway.

### **3.13 References**

1. The KNOWLEDGE GRID Lab. <http://dns2.icar.cnr.it/kgrid/>.
2. M. Cannataro and D. Talia. The Knowledge Grid. *Communications of the ACM*, 46-1, 2003.
3. A. Pugliese and D. Talia. Application-oriented scheduling in the KNOWLEDGE GRID: a model and architecture. *International Conference on Computational Science and its Applications (ICCSA)*, 2004.

4. M. Cannataro, A. Congiusta, A. Pugliese, D. Talia, P. Trunfio. Distributed data mining on Grids: services, tools, and applications. *IEEE Transactions on Systems, Man, and Cybernetics: Part B (TSMC-B)*. To appear.





## 4 GRASP (Grid Resource Allocation Services Package)

### 4.1 Summary

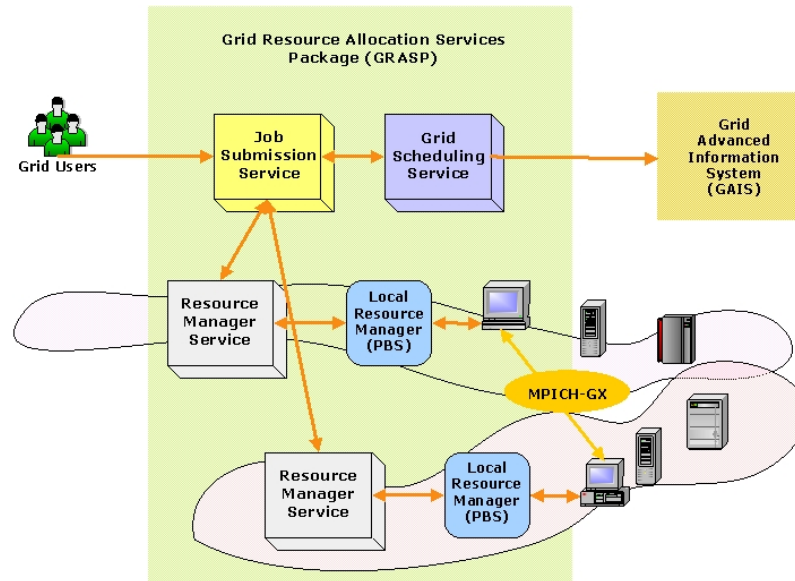
GRASP(Grid Resource Allocation Services Package) is designed to meet the requirements of the resource management problem concerning about delivering the users plentiful computing power with distributed resources. Currently, Managed Job Service in Globus Toolkit 3 is the service to be used to run the job on a remote resource. However, in order to build more useful grid, there should be added some user-friendly resource allocation manners including resource brokering, scheduling, monitoring, and so forth in the collective layer. GRASP is aiming at this upper-GRAM level scheduling and job submission system. Followings are brief introduction of GRASP functions.

- **Grid Job Submission:** GRASP has a service, Job Submission Service, where users are interfacing with grid computing environment. We solved the co-allocation problem for a cross-resource MPI-based parallel job by designing an MPICH initialization process in which all MPI subjobs are synchronized by Job Submission Service. And also monitoring in the service allows the user to monitor his/her job as a whole.

- **Resource Brokering and Meta-Scheduling:** A Grid Scheduling Service finds resources fit to a user's job derived from a grid information service. To select proper resources it performs matchmaking between a resource specification from the user and resource owner policies about jobs or users from each resource administrator. And then it selects resources to be allocated to the job from the candidates which have been found.

- **Local Job Execution :** A Resource Manager Service authenticates the user for the job execution on a local resource and submits the job to the local batch queuing system such as PBS. And this service will support the immediate reservation to minimize the failure of execution of scheduled job in the upper layer during meta-scheduling.

- **Fault Tolerant Job Execution :** Grids consist of so many computing resource components and each has a probability of local failure, which decreases the reliability of the whole grid system. To increase the reliability of the system, fault tolerance for a grid job is required. Without fault tolerance, parallel or distributed processes are vulnerable even at local single failure and might loose all computation mid-result on failure only to start from the beginning. We realized a fault tolerant job execution which makes a grid job restarted automatically from where the failure occurs, adopting the periodic checkpointing mechanism.



**Figure 4-1: Architecture of GRASP**

## 4.2 Customers

The target customers of GRASP would be mostly computational scientists who used to run parallel jobs in a grid environment.

## 4.3 Scenarios

### ○ Job Submission

Two major application types are considered: high throughput computing and high performance computing applications. In the case of high throughput computing, it is not necessary for each process on the resources to communicate with each other. Sensitivity analysis and parameter tuning studies are performed by high throughput computing method. The other application is MPI-based parallel job for the high performance computing, which requires significant amount of communications among the subjob processes. And also a hybrid of HTC and HPC, that is, a HTC job whose subjobs are MPI-based HPC jobs can be handled.

In order to support these kinds of applications, GRASP interprets user's job request, and then finds out and selects resources to appropriately run a job. After the scheduling process, the job is distributed to selected resources.

### ○ Job Restart on Failure

In GRASP the MPI-based job can resume its computation automatically even when the job stops because of the failure of any subjob process. There could be two kinds of failures on a distributed parallel job. One is a failure of a subjob process running on a computing node, the other is a failure of a resource on which subjob processes are running. When a process stops for its own reason, the cluster manager on the front node will fork a new process on that computing node. If the node is down, the cluster manager will choose another computing node in the cluster and resume the process on the node. More seriously if the whole cluster which the cluster manager is running on is down or

the connection to the cluster is lost, the central manager in the Job Submission Service will choose another appropriate cluster and resume the subjobs on the cluster. As mentioned above, GRASP takes hierarchical failure recovery system in which each failure manager handles the failures on each layer respectively.

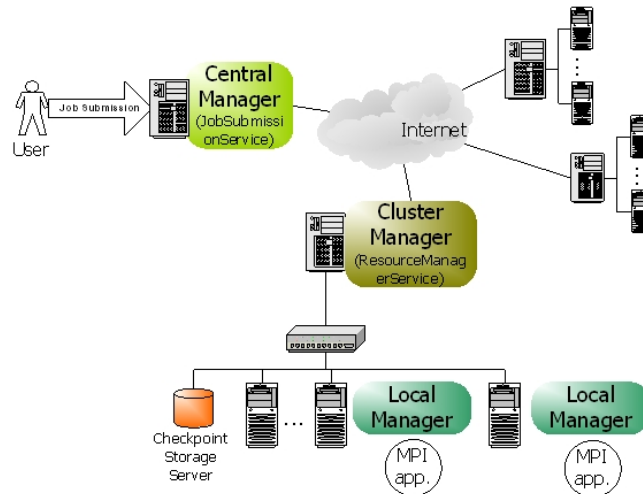


Figure 4-2: Fault Tolerance Job Execution Architecture

#### 4.4 Involved resources

For now, we are restricting resources as computing resources which are mainly clusters. However, we will gradually enlarge the scope of resources to storage devices, network connections, and so on.

#### 4.5 Functional requirements

##### ○ Discovery and Brokering

For dynamic resource offering and user convenience, a grid scheduling service should be able to discover and select proper resources from grid environment. In this phase, the scheduling service would be aided from an information service.

##### ○ Queuing

Grid environment is so dynamic and unpredictable that a grid job should wait in the queue until the scheduling process ends.

##### ○ Scheduling

Scheduling is a process of matching a job to the appropriate resources. In this phase, various scheduling algorithm could be applied.

##### ○ Authentication and Authorization

Authentication and authorization is essential to run a job on a remote resource. Therefore the scheduling component should check if the user can acquire the admission to the resources.

##### ○ Advance Reservation

Although the advance reservation is required for grid scheduling, we have not reached it yet. Therefore we are working on enabling the immediate reservation mechanism, in

which the scheduling service can occupy the resources at the time when the meta-scheduling is done.

- **Monitoring**

Monitoring job status and resource status could be considered. Job monitoring should be supported in the grid scheduling components. Although resource monitoring is required to discover resources, it would be supported by an information service.

- **Fault Tolerance**

Fault tolerant job management can make a grid system more effective because without a fault tolerance, the computation results upon the job failure would be blown up and the job should start again from very first step.

#### **4.6 Workflow of Scheduling Process**

1. Queue the job for the scheduling
2. Gather information about available resources from an information service
3. Filter unsuitable resources using matchmaking between the job specification and the resource owner policies which should be offered by the information service
4. Select the resources and number of nodes using various scheduling algorithm
5. Reserve the resources based on the schedule
6. Generate subjob request scripts for each resource
7. Submit each subjob request to the resources
8. Authenticate and authorize the user on the resources
9. Verify the reservations
10. Stage the required files on the resources
11. Execute subjobs on the computing nodes

#### **4.7 Involved Scheduling Components/Services**

- **Job Submission Service**

JSS(Job Submission Service) is responsible for management of the job. It receives a job request from clients, requests scheduling to a grid scheduling service, requests job execution to local resource management services, and controls the jobs during execution with the job monitoring.

GRASP supports an extended MPICH, which was implemented to make it possible for MPI subjobs that are dispersed on the remote resources to communicate each other. In this mechanism, JSS plays an important role to synchronize subjobs by controlling the barriers in each subjob process when an application is initialized.

And also, JSS handles job failure as a central manager of fault tolerant job execution system. It synchronizes the checkpointing process on the resources with each other, and handles the failure on a resource level not on a computing node level.

- **Grid Scheduling Service**

GSS(Grid Scheduling Service) discovers the resources available and chooses best fit resources for the job. In order to filter unacceptable resources, GSS does matchmaking between resource specification in the job request and resource owner's preference in the resource owner policy. The resource owner policy is delivered from the resource by information service. Then the candidates selected from the matchmaking process enter

the scheduling process and the final winners are picked out based on the scheduling algorithm. One example of the scheduling algorithm in GSS is point-based algorithm, in which all the resources have the point following user's preferences and the resources with higher points are selected. The last process in GSS is the reservation onto the resources scheduled.

#### ○ **Resource Manager Service**

RMS(Resource Manager Service) takes a job request from outside and starts execution of the user program on the resource with some required functions, authentication, authorization, file staging, output and error streaming, local scheduler interfacing, and so forth. And also, during execution the job can be monitored and controlled by RMS. In addition to the basic function of job submission, RMS supports JSS in synchronization of MPI-based job and GSS in reservation of the resource.

### **4.8 Failure Considerations**

GRASP can handle the job failure situations so that distributed processes don't lose their computation mid-results. Our approach is to adopt the periodic checkpointing mechanism to decrease the loss of computation results. Checkpointing is an operation to store the state of a process into stable storage so that a process can resume its previous state at any time with the latest checkpoint file. In our system, hierarchical job managers in JSS and RMS monitor and control MPI processes, that is to say, cluster manager in RMS and central manager in JSS are responsible for detecting node/network/process failures and deciding consistent global recovery line. [Figure 4-2]

### **4.9 Security Considerations**

Security functionality of all services in GRASP is based on GSI generally. More precisely, RMS, the grid service of computing resource follows the authentication architecture of GT3 GRAM.

### **4.10 Accounting Considerations**

For accounting, the usage of computing resources should be measured correctly. This measuring is done by extracting information from local scheduler on matching local user account to the grid user from outside. The grid user account is represented by a distinguished name(DN) in the gridmap file.

### **4.11 Performance Considerations**

GSS applies a cache mechanism in order to make a good performance in fetching resource information. When discovering the resource information for a job, the local cache is searched for the resources satisfying the query at first. Only if the proper information could not be found in its local cache, GSS make a query to an information service outside. The local cache is updated when new information reached from an information service, and updated by the Cache Auto Updater periodically using the notification mechanism in OGSI.

#### **4.12 Use case Situation Analysis**

GRASP is ongoing architecture to support scientific applications in the grid infrastructure. The implementation is not deployed in the real environment yet, but we are working on the deployment of GRASP in the Korean grid infrastructure. The first targets would be applications from bio-informatics, computational fluid dynamics using genetic algorithms, and some data-intensive applications.

#### **4.13 References**

[1] MoreDream Project, <http://www.moredream.org>

[2] Globus Project, <http://www.globus.org>

## 5 Editor Information

Philipp Wieder  
Research Centre Jülich  
Central Institute for Applied Mathematics  
52425 Jülich, Germany  
[ph.wieder@fz-juelich.de](mailto:ph.wieder@fz-juelich.de)

Ramin Yahyapour  
Computer Engineering Institute  
University Dortmund  
44221 Dortmund, Germany  
[Ramin.yahyapour@udo.edu](mailto:Ramin.yahyapour@udo.edu)

Andrea Pugliese, Domenico Talia  
DEIS-University of Calabria  
Via P. Bucci, 41/C, Rende ,Italy  
[{apugliese,talia}@deis.unical.it](mailto:{apugliese,talia}@deis.unical.it)

Jaegyeon Hahm  
Supercomputing Center  
Korea Institute of Science and Technology Information  
305-333, Daejeon, Korea  
Phone: +82-42-869-0580,  
[jaehahm@kisti.re.kr](mailto:jaehahm@kisti.re.kr)