Architecture of the Resource Management System of WP4 (Fabric Management) - Version 2, Draft -

Thomas Röblitz, Florian Schintke, and Thorsten Schütt
{roeblitz,schintke,thorsten.schuett}@zib.de

Zuse-Institut-Berlin (ZIB)
Takustr. 7, 14195 Berlin
Germany

September 24, 2001

Abstract

We report on the proposed resource management system architecture and its interaction and integration into the fabric management architecture and the global DataGrid architecture.

1 Introduction

This document describes the architecture of the *Resource Management System* (RMS) for *Work Package 4 - Fabric Management* (WP4) of the European DataGrid[1] project.

The goal of the DataGrid project is the development of a middleware (i.e. software) that enables transparent access to geographically distributed resources [4]. Because DataGrid is intended to be used by many people performing extensive computations on huge datasets [1, 2] new techniques have to be developed to manage tens of thousands of compute nodes and auxiliary resources.

A DataGrid Compute Center consists of one or more clusters. Job submissions to these clusters may be managed by different batch systems, e.g. PBS [12], LSF [11], Condor [3], etc.

The aim of the RMS is to develop a middleware that provides transparent access to different batch systems, e.g. job submission and retrieving information, and to enhance their capabilities with advance reservation and co-allocation if necessary.

The remainder of this document is organized as follows. In Section 2 we define the purpose of the RMS and discuss requirements on it. The architecture and requirements on other components will be presented in Section 3 and Section 4 respectively. The Control API will be presented in Section 5. In Section 6 we list some open questions. We conclude our findings in Section 7, which will also outline further work. Appendix A contains a glossary of used terms. The functionality of the architecture is illustrated by use cases presented in Appendix B.

2 Scope and Requirements

DataGrid wants to provide computing and storage services to a large user community. This service and its usage can be seen as a service of a *Virtual Organizations* as described in [6]. The resources of a compute center (e.g. clusters, network of workstations, etc.), which are fully or partially contributed to the *Virtual Organizations*, are managed by the *Fabric Management* (FM) of the DataGrid project. One (integral) part of the FM is the *Resource Management System* (RMS) which is described in this document. More information about resource management in general can be found in [9, 8, 10]. Its purpose is:

- to add/delete resources to/from the pool of man-

[1]The DataGrid project is funded by the EU and coordinated by CERN, the European Organization for Nuclear Research.

aged resources based on decisions made by the monitoring system, fault detections system, or by administrators,

- to handle resource requests from the WP1 (Workload Management),
- to handle resource requests from local users of a Compute Center,
- to schedule all jobs,
- to offer several scheduling strategies like *First Come First Serve* (FCFS), *Backfill*, *Shortest Job First*, *Longest Job First*, *Deadline Scheduling* or *Advance Reservation*,
- to perform accounting.

Jobs are described in the *Job Description Language* (JDL). We provide requirements that the RMS has on the JDL in Section 3.11.

The architecture of RMS has to meet several requirements which are imposed by general goals, grid services and other fabric management components. These are discussed in more detail in the following sections.

2.1 General Goals

Because of the huge number of resources to manage the RMS has to meet some general requirements to achive the general goals. These goals are:

Scalability: Applied mechanisms for scheduling, coallocation and information provision must scale to tens of thousands of resources.

Automation: To ensure a high quality of provided services and to reduce cost of ownership administration procedures, e.g. software installation/upgrading or fault recovery, must be highly automated.

Extensibility: Because of the intended long term use of the developed system, it is necessary that the system may be adapted and/or extended easily in order to support future technologies.

2.2 Requirements of Grid Services

This Section describes requirements imposed by other work packages.

2.2.1 Workload Management (WP1)

The work package 1 is responsible for workload management on the Grid level, e.g. accepting job requests from users, selecting appropriate resources, issuing job requests to these, etc. Jobs are described in the *Job Description Language* (JDL). The RMS can handle jobs described in the JDL.

WP1 needs information about available resources to distribute the workload to available resources. It is the task of the RMS to provide these information, which are published within a common framework the *Globus Metacomputing Directory Service 2* (MDS-2).

2.3 Requirements of other Fabric Management Subsystems

The RMS is a subsystem of the *Work Package 4 - Fabric Management* (WP4). Of course, the RMS closely interacts with components developed by the WP4. The requirements imposed by these subsystems[2] are discussed below.

Configuration Management: The Configuration Management stores the static configuration of the RMS subsystem. See requirements of the Software Installation task on JDL.

Fabric Information Service: The RMS publishes information via the Fabric Information Service. This is done periodically or on request.

Fault Detection and Recovery System: See requirements of the Software Installation task on JDL.

Gridification: The gridification task provides methods to access fabric services by grid services and vice versa. The *Local Credential and Authorization Service* (LCAS) performs authentication and authorization checks with a request received from the ComputeElement. It provides hooks to perform customized authorization checks. The RMS should provide modules (plugins) that perform accounting and quota checks and dynamic checking for resource availability.

The *Local Credential MAPping Service* (LCMAPS) provides all local credentials needed

[2]Note: The current architecture of WP4 does not plan a Fabric Information Service, even though we think that it could be quite useful to have such a thing.

by jobs allowed within the fabric. The RMS obtains all necessary credentials from the LCMAPS and should not schedule or start a job before having received all necessary credentials. The LCMAPS will be notified directly if a job has been scheduled or started and indirectly, via transmitting the result if the job has been finished.

The *ComputingElement* (CE) mediates a job request received from any grid entity to the RMS, i.e. the RMS accepts job requests from the CE and sends job results back to it. Both the RMS and the CE use a job repository for storing job related information.

The *FabNAT* will provide a method for streaming connections between nodes within a fabric and external nodes. It will assign ports or port ranges to jobs. Ports and port ranges have to be considered by the RMS as resources too.

Monitoring System: See requirements on JDL by the Software Installation task.

Software Installation: The JDL must allow to specify nodes by name or some address. This is necessary to perform 'administrative actions' on specific nodes.

3 Architecture

In Section 2 we listed requirements that influence the architecture of the RMS. Of course, the architecture of the RMS makes some assumptions about the environment. First, we discuss these assumptions, then we assume that these assumtions are true and continue by presenting the architecture in detail.

3.1 Assumptions of the RMS

The RMS is build on top of local batch systems, which can be configured in two different ways. First, a local batch system may be configured *directly*, i.e. as if the RMS would not exist. Second, a local batch system may be configured *indirectly* by using the Configuration Management of WP4, i.e. all settings are stored within the ConfigDB and then used to configure the local batch system. The RMS supports both possibilities.

The Configuration Management subsystem stores the static configuration information of the RMS (not only the information regarding the local batch systems). This information may be accessed either directly via an API proposed by the CM task or indirectly via the RMS Information System.

3.2 Overview of the Architecture

As described in Section 2 the tasks of the RMS are the handling of jobs and the provision of information about local resources and jobs. This is reflected with the architecture of the resource management system as shown in Figure 1. In this figure all components in dotted lines are provided by the RMS task. Components that are outside dotted line are provided by other tasks. The main RMS components are the *RMS Information Service*, the *Scheduler*, the *Proxies*, the *Request Handler* and the *Request Checker*. The RMS also provides methods to perform authorization checks, e.g. a plugin for *Local Accounting* deployed by the LCAS, a plugin to test for the availability of resources, and the provision of information, e.g. *Information Providers* deployed by the GriFIS.

3.3 RMS Information System

The RMS uses static and dynamic information. These information describe the states of the RMS and its managed resources. Static information, like the chracteristics of nodes, are stored by the Configuration Management subsystem. Dynamic information may be obtained from local batch systems, the scheduler or the monitoring subsystem. All these information are accessed by components of the RMS and other subsystems of WP4. The access may be organised in different ways.

The individual information access is shown in Fig. 2. Each component obtains its desired information directly from the information producer. The *RMS Information Repository Manager* (RMSIRM) is responsible for controlling access to the repository by components of the RMS. An advantage of this method is its high efficiency of information exchange. A big disadvantage is that every component must interface with many other components and these interfaces may differ or not implement a common protocol.

The uniform information access is shown in Fig. 3. All components access information through another component, the *RMS Information Repository Manager* (RMSIRM) or via a common interface. A repository stores all information that is needed by the RMS. This includes information generated by the RMS itself, e.g. a

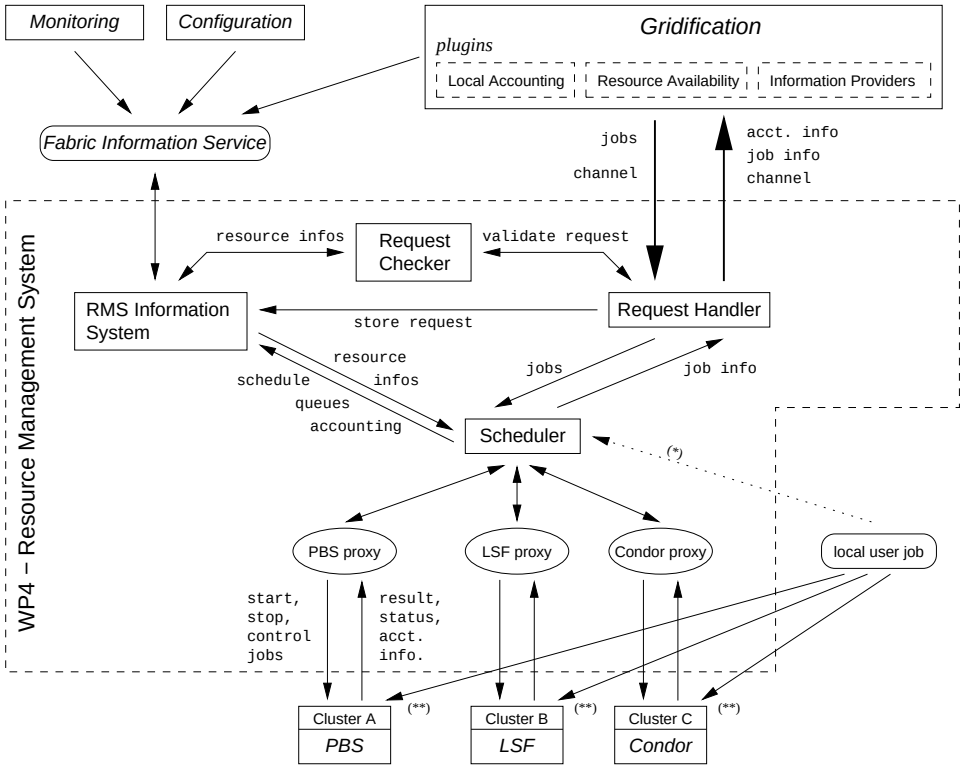

Figure 1: Architecture of the resource management system of WP4. (*) Submission of local user jobs as described in cases 1a and 1b. (**) Submission of local user jobs as described in cases 2a, 2b and 2c. (See discussion in section 'Local Batch Jobs'.)

schedule, and cached information of other components, e.g. configuration information. The RMSIRM is responsible for controlling access to the repository. Information needed by other components of WP4 are published to the Fabric Information Service via the *RMS Information Publisher*. The *Fabric Information Caching Manager* provides access to the Fabric Information Service (e.g. Configuration Information or Monitoring Information) and manages cached information pieces, that are temporarily stored in the RMS Information Repository. An advantage is the common interface. A disadvantage may be a reduced effciency of information exchange, because of using indirect accessing methods. Fig. 3 also shows the interplay of the RMSIS with the Fabric Information Service. If this component will not be implemented, the Fabric Information Caching Manager and the RMS Information Publisher have to communicate directly with the other components, e.g. Configuration Database, Monitoring, etc.

3.3.1 Information Schemata

The RMS Information System stores its information in a database. This could be just a collection of files (one per schema) or a relational database. Information about entities are combined within logical structs, described by schemata. Here we will give an overview on the most important schemata and their attributes. In Table 1, Table 2 and Table 3 we present schemata for node, queue and job information respectively.

3.4 Request Handler

The Request Handler accepts requests delivered via the CE of the *Gridification task*. After receiving a request it manages it by going through the following steps:

1. Store the request into the RMS Information System.
2. Verify the request, e.g. certified by LCAS, basic check for resource requirements, etc. This is done

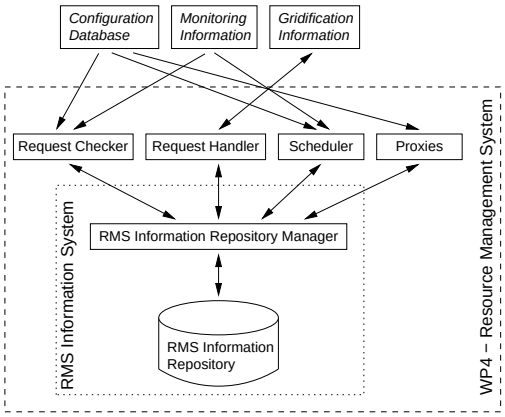

Figure 2: Detailed architecture of the RMS Information System: *Individual information access*.

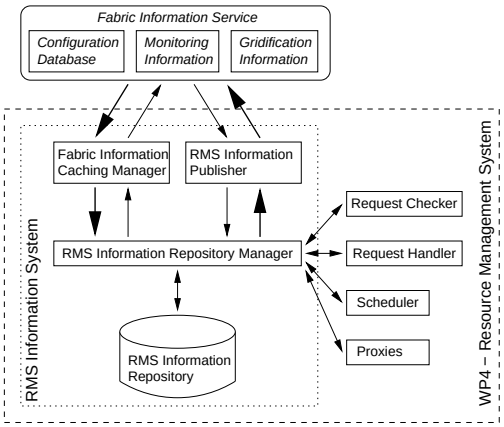

Figure 3: Detailed architecture of the RMS Information System: *Uniform information access*.

by the *Request Checker* using information from the Configuration Management.

3. Obtain all local credentials from the Gridification subsystem.

4. Schedule the job, i.e. match the resource requirements, advance reservation, co-allocation and inform the CE.

5. When the job should start, recheck the job requirements with the current status of the local batch systems. This is done with information form the Monitoring System. If this is not successful, try to reschedule the job or cancel the job with a 'FAILED' message.

Table 1: Schema for node information.

Attribute	**Description**
`NodeId`	A number that represents the node, e.g. MAC address.
`Architecture`	The architecture of the CPU(s).
`OpSys`	The operating system of the node.
`Processors`	The number of CPUs installed on this node.
`Status`	The current status of the node, e.g. idle, running, maintenance, etc.

6. Submit the job to the local batch system.

7. Update status information of the job during its execution (e.g. transmit these information to the CE) and wait until the job has finished.

8. Store status information of the job into the RMS Information System too.

9. Send these information back to the CE. It should contain job status and accounting information.

If the verification or scheduling step fails, the Request Handler replies with a 'FAILED' message. This message may explain the reason, but it may also hide it due to site policies.

3.5 Request Checker

The Request Checker verifies several characteristics of a job request. It may check the existence and integrity of the LCAS certificate. Furthermore, it verifies whether the fabric can provide the requested resources.

3.6 Scheduler

The task of the scheduler is it to assign resources to incoming job requests. It may deploy different strategies to generate a schedule, that fulfills the job requirements, makes efficient use of available resources and applies site policies. First, we describe how the scheduler works in more general. Then, we discuss how the scheduler interfaces with the local batch systems. Last, we explain how maintenance jobs/events may be taken into account.

Table 2: Schema for queue information.

Attribute	Description
Name	The name of the queue.
Architecture	The architecture of the machines associated to this queue.
OpSys	The operating system of the machines associated to this queue.
TotalCpus	The total number of CPUs associated to this queue.
FreeCpus	The number of free CPUs being able to run jobs submitted to this queue.
TotalJobs	The total number of jobs in the queue.
RunningJobs	The number of jobs currently running in the queue.
IdleJobs	The number of jobs not running in the queue.
MaxTotalJobs	The maximum number of jobs allowed for this queue.
MaxRunningJobs	The maximum number of running jobs allowed for this queue.
Status	The status of this queue.
RunWindows	The time windows that define when the queue is active.
Priority	The priority of this queue.
MaxCpuTime	The maximum CPU time allowed for jobs submitted to this queue.
MaxWallTime	The maximum wall clock time allowed for jobs submitted to this queue.

Table 3: Schema for job information.

Attribute	Description
GlobalJobId	The Globus id of the job.
LocalJobId	The id of the job in the underlying resource management system.
GlobalUser	The id of the Grid user.
LocalUser	The username in the local system.
Status	The status of the job.
PendingReason	The reason for which the job is not running.
RSLCommand	The RSL string used to submit the job.
SubmitTime	The time at which the job has been submitted.
StartTime	The time at which the job first began running.
WallClockTime	The wall clock time accounted for this job.

How does the scheduler work?

First, the scheduler matches the resource requirements described by the job request with the existing resources. If this process was successful, the scheduler makes provisional assignments of the resources to the job, i.e. it generates or updates a schedule. The schedule is provisional in the sense, that the assignment to actual resources is done just before the scheduler submits jobs to the underlying batch systems. An advantage of this behavior is that the scheduler can perform better load balancing, adapt the schedule to resource failures and consider maintenance events.

Local Batch Jobs

In this paragraph we will discuss how jobs are handled, that are submitted to local batch/queuing systems. Finally we will see that there should be only one component responsible for job scheduling. This implies that no extra component to schedule 'administrative tasks' should be used.

One requirement for RMS is to integrate clusters that are managed by local batch systems without affecting the required behavior of local users to submit jobs too much. The following paragraph summarizes how local and grid jobs may be handled. We distinguish some possible cases and discuss them afterwards. The cases are:

1. All jobs are submitted to local batch systems via the scheduler.
   (a) A local user submits her job **directly** to the scheduler.
   (b) A local user submits her job by using a **wrapper** that provides a common environment for the user.
2. Jobs of local users are submitted directly to the local batch systems.
   (a) The scheduler suspends the local jobs remotely, but they are kept in the queue of the local batch system, to allow fair execution of grid jobs. Grid jobs are submitted to the local batch systems just before they start. The

scheduler releases jobs of local users in respect of its schedule.

(b) Jobs are queued, but will not be executed. The scheduler generates a plan considering these jobs. If a job reaches its execution time, it is moved from its queue to an execution queue. Grid jobs are inserted into this queue just before they start. Only the scheduler may insert jobs into the execution queue.

(c) The scheduler does not suspend jobs in local batch systems. Grid jobs are submitted just before they start.

   i. The scheduler does not do any reordering of jobs within a batch queue.

   ii. The scheduler may reorder requests within a batch queue.

We will discuss these possibilities now regarding several arising issues, which are *fairness*, *advance reservation*, *security*, *job status* and *load balance*.

Fairness. By the term fairness we mean that both kind of jobs, issued by either local or grid users, are equally executed according to site quotas. In Figure 4 we illustrate two cases where fairness matters. The architecture should provide efficient solutions for these cases.

Figure 4(a) shows an example of inefficient use of resources. When a local batch system uses FIFO scheduling for a queue, it may happen that a job (`job C`) is delayed after another one (`job B`), even if the required resources are available and it will be finished before the other one (`job B`). A solution to this problem would be either using a more appropriate scheduling algorithm at the level of the local batch system or letting the RMS Scheduler be the only place where scheduling happens (cases 1a, 1b and 2b).

Figure 4(b) shows a race condition between local and grid jobs. The local batch system uses a FIFO scheduler for a queue. The RMS Scheduler had make a provisional assignment for job B and verifies the queue status just before submitting the job to the local batch system. However a local user submitted job C which uses resources that are needed for job B. In this case the submission of job B failed and the RMS Scheduler has to reschedule it. A solution for this problem would be either performing the status lookup and job submission as an atomic action or letting the RMS Scheduler be the only place where scheduling happens. If the first solution can be achieved depends on the underlying batch system. Furthermore, one must enable atomic actions for each different batch system and no guarantee can be given if these systems provide upwards compatibility with newer versions. Thus the second solution, only the RMS Scheduler does the scheduling (cases 1a, 1b and 2b) is preferred.

Advance Reservation. The concept of *Advance Reservation* enhances the capabilities of the RMS Scheduler and provides an additional feature for users who are planning huge jobs or presentations. However, local batch systems may not provide this feature, but the RMS Scheduler may implement it. Submitting local user jobs directly to a local batch system (cases 2a and 2c) may invalidate the schedule of the RMS consecutively. Thus advance reservation requires that local user jobs are submitted via the RMS Scheduler or that the RMS Scheduler deploys case 2b.

Security. The RMS has to be protected against abusing its services, especially the unauthorized submission of jobs. The gridification task ensures this requirement for grid jobs. If local user jobs are submitted to a local batch system directly (case 2a and 2c), security is not a concern of the RMS. If local user jobs are submitted via the RMS Scheduler, the architecture must ensure that jobs may not be submitted to local batch systems directly. This can be achieved by restricting the access to the local batch systems, i.e. only the RMS Scheduler may submit jobs to a local batch system or by using an addtional execution queue as in case 2b.

Job Status. Local batch systems offer several opportunities to lookup the status of managed resources, e.g. which jobs are submitted to which queue and what is their status. Because the RMS Scheduler submits jobs to the local batch systems just before their starting time, local users may not see all jobs assigned to a queue. Wrapping query methods may help, but local users may not use these. Thus, they can be confused if they neither see a job submission failure message nor their jobs queued to the chosen resources. There can be no unique solution to this problem. When a job has been scheduled successfully, the RMS supplies information to the user how to query the desired information, i.e. which wrapped method should be used.

Load Balance. If more than one cluster is used in a compute center and jobs are submitted directly to the

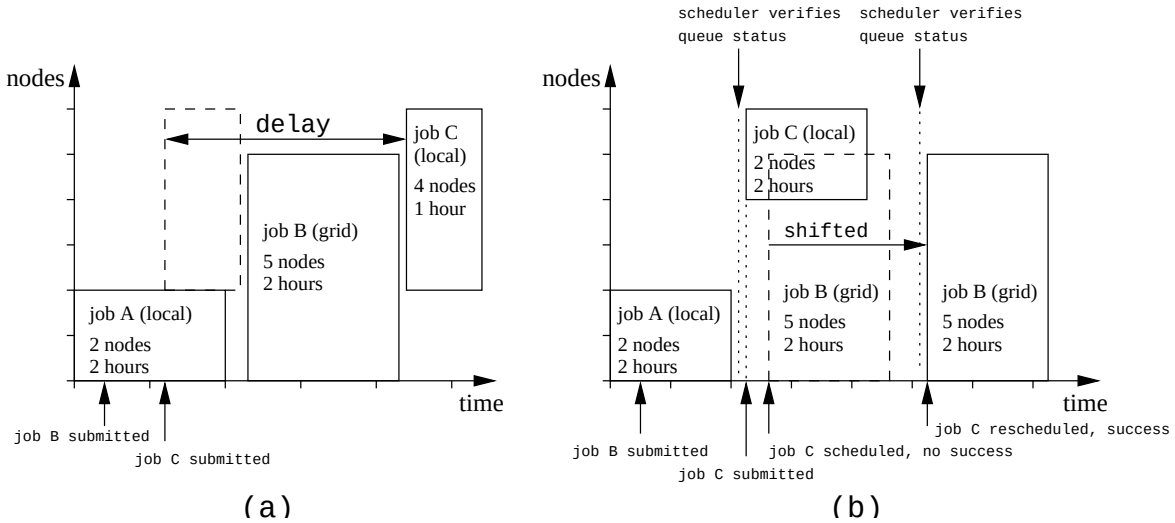

Figure 4: Situations that may happen if local and grid jobs are not scheduled via a single component.

local batch systems, it is difficult to provide good load balancing over the clusters. If one uses a central schedule we can decide on availability issues to which cluster we want to submit a job immediately before its execution and can better respect short term issues. To delete a job from one local batch system (PBS) and to submit it to another (Condor) seems to be impractical.

Conclusion. From the discussion above we concluded that case 1b or 2b are the best to implement. Because, case 2b does interface the local batch systems better, especially regarding security and the easiness of use by local users, it may be choosen as first approach.

Maintenance Jobs

Besides user jobs (grid or local) there also exist maintenance jobs, i.e. applying configuration changes, recovering after a detected failure, etc. Maintenance jobs are dedicated to specific nodes, i.e. the affected nodes are explicitly given in the job request. We assume that maintenance jobs cannot be executed in parallel with user jobs. Otherwise, maintenance jobs could affect the performance of a node and thus invalidate the job requirements. A more important reason is that actions like rebooting the node or performing a complete reinstallation of a node could be part of a maintenance job as well.

Of course, this strict handling of maintenance jobs may not be necessary for all kind of configuration changes. It may well be that maintenance jobs are devided into *intrusive* ones, i.e. which may not be executed in parallel with user jobs, and *non-intrusive* ones, i.e. which may be executed in parallel with user jobs. This is kept open for the future, because it also needs some experience on how to classify maintenance jobs as intrusive or non-intrusive.

If a maintenance job should be executed, the initiating component (e.g. the Fault Recovery System) asks the RMS Scheduler for a time slot on the affected nodes. It should be possible to estimate the duration of a maintenance job in advance. Thus the RMS Scheduler can include maintenance jobs in the schedule. It may also respect the affected nodes for scheduling of jobs for the time after the maintenance will have been finished. But this has to be done carefully, because the configuration of the nodes, e.g. installed software, may have been changed.

3.7 Proxies

Proxies enable the RMS Scheduler to access local batch systems. Each local batch system will be accessed via a distinct proxy. A proxy consists of one or more scripts. These scripts translate job descriptions in *Condor ClassAds* into commands and parameters of the local batch system or translate job results into a format used by the RMS Scheduler.

3.8 Plugin for Local Accounting

The *Gridification task* performs a set of authorization checks. These checks can be implemented by modules which are called by the LCAS component. The RMS provides a module that verifies accounting information. The module needs local information, which is stored in the Fabric Information Service or the RMS Information System, and global information, which is managed by a central service.

3.9 Plugin for Resource Availability Checks

The RMS provides a module that checks the availability of resources. This module will be deployed by the Gridification task like the plugin for local accounting.

3.10 Information Providers

The workload management (WP1) needs information about available resources at a Compute Center. These information are stored using MDS-2. An *Information Provider* is a script that delivers information about a resource, e.g. Queue or ComputingElement.

3.11 Job Description Language

The Job Description Language (JDL) is used to describe jobs, e.g. the name of the executable, input and output data, resource requirements, etc. As agreed with WP1 (Workload Management) the JDL will be based on *Condor ClassAds* for PM9. Later on the JDL syntax may be switched to XML in order to reduce the need to develop parsers to parse the JDL.

In Table 4 we present a few additional attributes which may be valuable to describe a job.

Table 4: Additional attributes for the JDL.

Attribute	Description
`JobDependencies`	It is possible that several jobs must be finished before this job can be started. The value of this attribute can be a list of GlobalJobIds.
`PhysicalNode`	The physical node number of the node where the job has to be executed (necessary for maintenance/administrative jobs).
`TimeOut`	The RMS tries to get the requested resources within the specified timeout. If this is not possible, a 'job failed' notification will be send back to the job submitter.
`JobSubmitter`	To have a backwards communication channel for diagnostic information (job fail, ...).

4 Requirements on other Fabric Management Tasks

Here we describe the requirements that the RMS has on other Fabric Management Tasks. Some of them was already discussed in the document and are repeated here for reference. Maybe this list is not complete. If you find some implicit requirements on other resource management tasks in this document that are not listed here, please contact one of the authors (Thomas Röblitz, Florian Schintke, or Thorsten Schütt {roeblitz, schintke, thorsten.schuett}@zib.de).

Configuration Management:

- Configuration changes made by local administrators must be delivered to the RMS via the Fabric Information Service or an equivalent component with a push method.

Fabric Information Service:

- Components may register callback routines, which are called, when certain events, e.g. change of the configuration, happened.

Fault Detection and Recovery System:

- If some nodes has to be eliminated from clusters or has to be reintegrated this information has to be delivered to the RMS, e.g. by using methods of the control API of the RMS.

Gridification:

- Jobs in JDL
- Jobs are checked via LCAS
- Interface to provide information about a job to the grid.
- Local credentials are supplied by LCMAPS.

Monitoring System:

- Provides information about resources, so that job requirements can be checked to be still satisfied immediately before the execution.

5 Control API

The Control API defines the interface of the RMS. Other subsystems may use the primitives to trigger actions or set the state of the RMS and its managed resources. Below, we will present the API by naming the primitives and their parameters and return values.

getTimeSlot - to obtain a time slot on a node or some nodes, e.g. for maintenance jobs
Parameters: node/s, begin, end, duration, priority
Return value: result, per node (begin,end)

announceNode - to declare a node or some nodes available for production
Parameters: node/s
Return value: result

removeNode - to remove a node or some nodes from production
Parameters: node/s, when
Return value: result

setNodeState - to set the state of a node or some nodes
Parameters: node/s, state/s
Return value: result

getNodeState - to get the state of a node or some nodes
Parameters: node/sstate/s
Return value: state/s

announceQueue - to declare a queue available for queuing
Parameters: name
Return value: result

removeQueue - to remove a queue
Parameters: name
Return value: result

setQueueParameters - to set parameters of a queue
Parameters: name, attributes of the queue
Return value: result

getQueueParameters - to get parameters of a queue
Parameters: name
Return value: attributes of the queue

getQueueState - to set the state of a queue
Parameters: name, state
Return value: result

getSchedule - to obtain the schedule
Parameters: [jobid], [queue]
Return value: schedule

waitForFreeNode - to wait until the node is free, i.e. not executing a job nor will execute a job
Parameters: node, waitTimeout, [duration], force
Return value: result

releaseNode - to release a node, usually called in conjunction with **waitForFreeNode**
Parameters: node
Return value: result

announceConfigChange - to announce a configuration change, a parameter may specify when the configuration will be updated (by jobid or date)
Parameters: entity, [jobid], [date]
Return value: result

6 Open Questions

Open questions are distributed over this document. Open questions that aren't related to discussed aspects in this document are collected here. Note: Some open questions can be found in the Appendix.

- *Which local batch systems support scheduling of jobs to specific nodes and how?*

- *How do the fabric components communicate occuring events with eachother? Is a general event service provided by the fabric information service a proper solution? (pushing instead of polling information)*

7 Conclusion and future work

The main aspects of the resource management system was discussed in this document. Requirements from and to other tasks have to be discussed in more detail. Adjustments of the architecture will be done based on these discussions.

A Glossary

Administrator: A human resource that administers resources.

Advance Reservation: A reservation done for a time interval in the future, to run jobs.

API: ↑ Application Programming Interface.

Application Programming Interface (API): definition of the interaction between a program that provides an interface with other programs.

Batch Mode: A scheme where jobs are submitted and executed at a later point in time. ↑ Interactive Mode.

CE: Computing Element

Co-allocation: The allocation of more than one resource of different kind or different management system at the same time to use them concurrently (network and computing resources or one cluster and another cluster).

Compute Center: An institution that provides resources to the grid.

Condor: A High-Throughput Computing system that submits jobs to idle workstations [3].

Condor ClassAds: A language that is used for describing jobs, their resource requirements and resources.

Condor proxy: A program or set of programs that mediates between the RMS Scheduler and Condor.

Configuration: An assignment between variables and values. The actual behaviour of the RMS components depends on how RMS specific variables have been set.

Daemon: A process that waits for service requests.

Fabric Information Service: An information service used exclusively by the fabric management components.

Fabric Management (FM): The management of a Compute Center (↑).

FM: ↑ Fabric Management

Globus: The Globus project is developing fundamental technologies needed to build computational grids [7, 5].

Grid User: ↑ Local User

Information Provider: A program that provides information about local batch systems. The information is stored in a MDS-2 component.

Interactive Mode: A scheme where jobs are immediately executed when they are submitted. ↑ Batch Mode.

Job: A request to execute an application.

Job Description Language (JDL): A language that is used to describe a job.

JDL: ↑ Job Description Language.

LCAS: Local Credential and Authorization Service

LCMAPS: Local Credential MAPping Service

Local Accounting: The accounting performed on rules defined by local administrators.

Local Batch System: ↑ Local Resource Management System.

Local Queueing System: ↑ Local Resource Management System.

Local Resource Management System: A resource management system that is dedicated to a cluster.

Local User: ↑ Grid User

LRMS: ↑ Local Resource Management System.

LSF: Load Sharing Facility, a commercial batch system [11].

LSF proxy: A proxy that mediates between the RMS Scheduler and LSF.

MDS-2: The Globus Metacomputing Directory Service 2.

Migration: Moving jobs from one node to another.

Monitoring: Observing resource and job characteristics.

PBS: Portable Batch System [12].

PBS proxy: A proxy that mediates between the RMS Scheduler and PBS.

Process: A part of the execution of a job.

Proxy: A program or set of programs that mediates between the RMS Scheduler and local batch systems.

Request Checker: A component of the RMS that verifies request characteristics.

Request Handler: Receives request from the CE (↑) and coordinates the work of the RMS components.

Resource Definition Language: A language used to describe resources (↑ JDL).

Resource Management: ↑RMS

Resource status:

Resource: An entity managed by the RMS, e.g. nodes, queues, clusters, storage, etc.

RMS: Resource Management System is a set of components which work together.

RMS Information System: A component that stores information needed by the RMS components. Parts of the information may be retrieved from or published to the Fabric Information Service.

Schedule: A plan of assignments between jobs and resources.

Scheduler: A program that determines a schedule for jobs.

Task: ↑ Job

User Interface: An interface allowing an user to access the capabilities of a system.

User: A human being who is using a system.

WP1: Work Package 1 - Workload Management.

WP4: Work Package 4 - Fabric Management.

B Use Cases

B.1 Scheduling grid jobs

The RMS receives a job request from the CE. First, the Request Handler asks the Request Checker to verify the job request. The verification includes checking the authorization information that has been augmented to the request by the LCAS, and some resource requirements, e.g. if the fabric provides the requested resources, but neither schedule nor deadlines are considered. Next, the Request Handler asks the scheduler to process the request. The scheduler matches the job requirements to available resources and puts the job into its schedule if the matching succeeded. If the matching did not succeed, a failure message is sent back to the CE via the Request Handler.

When the starting time of the job has been reached, the scheduler repeats the matching of the job requirements and submits the job to the local batch systems if the matching succeeded. Last, the scheduler receives the results from the local batch systems and sends it back to the Request Handler. The Request Handler finishes the handling of the job by sending the result to the CE.

B.2 Scheduling local jobs

A local user submits a job to a specific queue of a cluster. The RMS Scheduler takes notice of this polling for the status of the queues or being signaled through a notification. Then, the scheduler matches the requirements and puts the job into its schedule. When the job reaches its start time, the RMS Scheduler moves it from the original queue to an execution queue that is only accessible by the RMS Scheduler. When the job has been finished, the scheduler sends a respond to the local user.

B.3 Node failure

If a node crashes due to some arbitrary reason, the scheduler must take this into account, i.e. for scheduling incoming job requests and for rescheduling already scheduled jobs. The Fault Detection System observes the failure, and triggers the Fault Recovery System to deal with the failure. The recovery procedures may involve the calling of one or more control methods of the RMS to perform maintenance jobs on the node. The scheduler may verify if its schedule is affected. The

RMS Information System may also be notified in order to update its information repository.

B.4 Configuration change

The configuration affects the scheduling of job requests. Thus, changes of the configuration should be delivered to the scheduler as soon as possible. If a configuration entry changes, an event will be generated. The scheduler receives the event and verifies if its schedule has to be adapted. The RMS Information System may also be notified in order to update its information repository.

References

[1] M. Aderholz, et al. Models of networked analysis at regional centres for LHC experiments, March 2000.

[2] S. Bethke, M. Calvetti, H.F. Hoffmann, D. Jacobs, M. Kasemann, and D. Linglin. Report of the steering group of the lhc computing review. Technical report, CERN European Origanization for Nuclear Research, February 2001.

[3] Condor Project Homepage. http://www.cs.wisc.edu/condor/, June 2001.

[4] EU DataGrid. http://www.datagrid.cnr.it, August 2001.

[5] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. 11(2):115–128, 1997.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabeling scalable virtual organizations, 2001.

[7] The Globus Project. http://www.globus.org/, June 2001.

[8] A. Keller, M. Brune, and A. Reinefeld. Resource Management for High-Performance PC Clusters. *Lecture Notes in Computer Science*, 1593:270–281, 1999.

[9] A. Keller and A. Reinefeld. CCS Resource Management in Networked HPC Systems. In *Proc. of Heterogenous Computing Workshop HCW'98 at IPPS, Orlando, 1998; IEEE Computer Society Press*, pages 44–56, 1998.

[10] Axel Keller and Alexander Reinefeld. Anatomy of a resource management system for HPC clusters. In *Annual Review of Scalable Computing*, volume 3. 2001.

[11] LSF Homepage. http://www.platform.com/products/LSF/, August 2001.

[12] OpenPBS Project Homepage. http://www.openpbs.org/, August 2001.