



# **GRASP**

## **(Grid Resource Allocation Services Package)**

### **User's Guide**

KMI-R1 Developer Team <kmi@moredream.org>

Document Version: 0.9     Date: November 07, 2005



Copyright 2002-2005 Korea Institute of Scientist and Technology Information.  
All rights reserved.

This document is licensed under the terms of the K\*Grid Public License.  
The details of K\*Grid Public License is found at  
<http://kmi.moredream.org/downloads/license.html>.

# Contents

1	Introduction.....	4
1.1	What is GRASP? .....	4
1.2	Architecture and Components of GRASP .....	4
1.2.1	Overview .....	4
1.2.2	JSS (Job Submission Service).....	7
1.2.3	GSS (Grid Scheduling Service) .....	8
1.2.4	RMS (Resource Manager Service) .....	10
1.2.5	RRS (Resource Reservation Service) .....	12
1.2.6	SRB enabled globus-url-copy .....	13
1.2.7	JRDL (Job & Resource Description Language).....	15
	Single job examples .....	30
1.2.8	Client Tools.....	34
1.3	Getting help .....	35
2	Installation and Configuration .....	36
2.1	Support software .....	36
2.1.1	Required.....	36
2.2	Installing support softwares .....	36
2.2.1	Installing Globus Toolkit.....	36
2.3	Installing GRASP Client.....	36
2.3.1	Client Tools.....	37
3	References.....	39
	Appendix A: A Client Program Providing Command Line Interface (CLI) : grasprun .....	40
A.1	Validate a JRDL .....	40
A.2	Job Submission.....	40
A.3	Job List and Information .....	42
A.4	Job Monitoring and Controlling .....	43

# 1 Introduction

This document is intended to provide the users with the information required to use GRASP.

## 1.1 What is GRASP?

The problem of Grid resource allocation is concerning about delivering the users distributed resources with computing powers, data storage capacity, network connectivity, etc. The Managed job service in Globus toolkit 3.x (GT3) 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 features and advanced resource allocation techniques including resource brokering, scheduling, job monitoring, and so forth. To meet this requirement in Grid resource management area, we designed and implemented a resource allocation system named GRASP(Grid Resource Allocation Services Package), which is to let users to submit their jobs in more efficient and intelligent manner to the Grid resources. The services of GRASP were implemented based on the OGSi specification implementation of GT3 as well as other services in MoreDream. Followings are brief introduction of GRASP functionalities.

## 1.2 Architecture and Components of GRASP

### 1.2.1 Overview

GRASP supports scientific applications with the high performance computing features such as MPI, high throughput computing features such as parametric studies and data intensive features. GRASP can handle three kinds of job type: SINGLE, XMPI, and HTC. SINGLE is a simple job to use only one node. XMPI is an MPI job which can be run over multiple resources. Lastly, HTC is a job for HTC applications like parametric study. In Both XMPI and HTC job case, GRASP co-allocates multiple resources to the job even though the resources are remotely distributed. To support data intensive features, we added the feature to automatically stage in files from SRB server and stage out the files to SRB server

Furthermore, we have designed the job description language, named JRDL (Job and Resource Description Language) to overcome the limitation of the GT3. RSL2, the GT3 job description language, just describes job specifications rather than resource specifications such as resource preference. RSL2 is also not considering about co-allocation. Therefore, we have proposed JRDL to meet both requirements for the job and resource's preference. The resource preference part is used in matchmaking step in Grid scheduling service (GSS). JRDL is designed based on XML schema.

GRASP is composed of four useful services needed to allocate the resources in Grid as illustrated in Figure 1. Firstly, the resource brokering is done by Grid scheduling service (GSS). This service finds out resources from the index service which are fit to a user's job and then reserve the resources in advance through Resource reservation service (RSS). To select proper resources it performs matchmaking between the resource specification from the user and the job/user specification preferred by the resource administrator. And then the resources are allocated to the job. Secondly, Job submission service (JSS) does co-allocation of resources and co-monitoring of the job. Co-allocation in GRASP makes it possible the job submission to the multiple distributed resources simultaneously. And co-monitoring allows the user to monitor her job flow. Lastly, Resource manager service (RMS) authenticates the user for the job execution on a local resource and submits the job to the local batch queuing system such as PBS. RMS should get the permission to allocate resource from RRS before submitting the job. Followings are the main features of GRASP, job types that is handled by GRASP and job statuses defined in GRASP. The explanation of each service, JSS (Job Submission Service), GSS (Grid Scheduling Service), RMS (Resource Manager Service), and RRS (Resource Reservation Service), will be followed after this overview section.

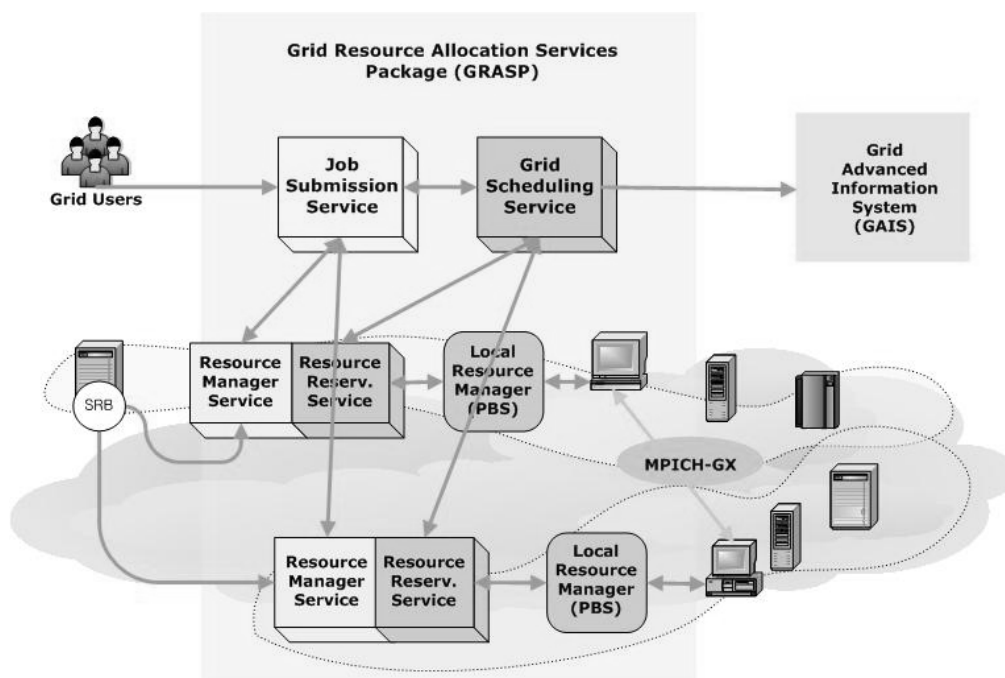


Figure 1. Architecture of GRASP

## Main Features

- All services are OGSI-compliant Grid services.
- GRASP supports three kinds of job type: SINGLE, XMPI, and HTC. SINGLE is a simple job which uses only one computing node. XMPI is an MPI job which can be run over multiple resources. Lastly, HTC is a job for high throughput computing such as parametric study.
- Multiple resources can be co-allocated to a job even though the resources are remotely distributed.
- Scheduler can automatically select resources by matchmaking process.
- Job can reserve resources in advance.
- The input files can be staged in from SRB server and the output files can be staged out to SRB server automatically.
- We provide JRDL (Job and Resource Description Language) as a general language to describe a job and user preferences required allocating resources for a job in Grid environment.
- We bring client tools for job creating, submission, controlling, and monitoring. They provide three user interfaces having same functionality: a command line interface, a graphic user interface, and web interface.

### **Job Type**

We are supporting three kinds of job type: SINGLE, XMPI, and HTC.

- SINGLE: It is a simple job which uses only one computational node (e.g. simple script for pre/post processing).
- XMPI: It is an MPI job which uses multiple resources to run even though resources are remotely distributed. Each resource could have several nodes.
- HTC: It is a job which uses multiple resources to run and have no communication between each of all subjobs (e.g. parametric study). Each subjob must be a SINGLE job.

### **Job Status**

#### Job Submission Status

Job submission service manages the status of job submission. The Status has following information:

- State of job
- All subjobs' statuses
- Fault message.

### Job State

- "Unsubmitted": JRDL is unsubmitted to Job submission service.
- "Scheduling": Job is scheduling to find proper resources at Grid scheduling service.
- "Pending": Job is pending even though the subjob is submitted to Resource manager service.
- "Active": Job is active.
- "Suspended": Job is suspended.
- "Done": Job is done.
- "Failed": Job is failed.

### Subjob Status

Resource manager service manages the statuses of subjobs. Each status has following information:

- Subjob id
- State of subjob state
- Execution time of subjob: start time and end time of job
- Allocation information of subjob: allocated resources' address
- Fault message.

### Subjob State

- "Unsubmitted": Subjob is unsubmitted to ResourceManagerService.
- "StageIn": Subjob is staging in the files to need to execute.
- "Waiting": Subjob is waiting for its requested execution time to be reached
- "Pending": Subjob is pending even though the subjob is submitted to the local job manager
- "XMPI\_init": XMPI subjob is initializing
- "Active": Subjob is active.
- "StageOut": Subjob is staging out the files to result from executing
- "Suspended": Subjob is suspended.
- "Done": Subjob is done.
- "Failed": Subjob is failed.

## 1.2.2 JSS (Job Submission Service)

### Key concepts

JSS is a Grid service to enable a job to submit to the resources in Grid testbed and enable a

user to monitor the status of submitted job. We provide JRDL language to describe the job, which is "an atomic task" of a workflow specification or other kinds of a complex, multi-step application. The service has the status of job submission and the requested JRDL which are provided as service data.

### Architecture

The job submission process is illustrated in Figure 2. When JSS receives the JRDL, the service can determine resources by GSS which is a Grid service to find out resources which are fit to the user's job from information provider and make a reservation to RRS on each resource. User could the resources manually by specifying the address and local job manager type of resources to JRDL. If the resources are decided, the job is divided into subjobs, then that are co-allocated to RMS on each resource.

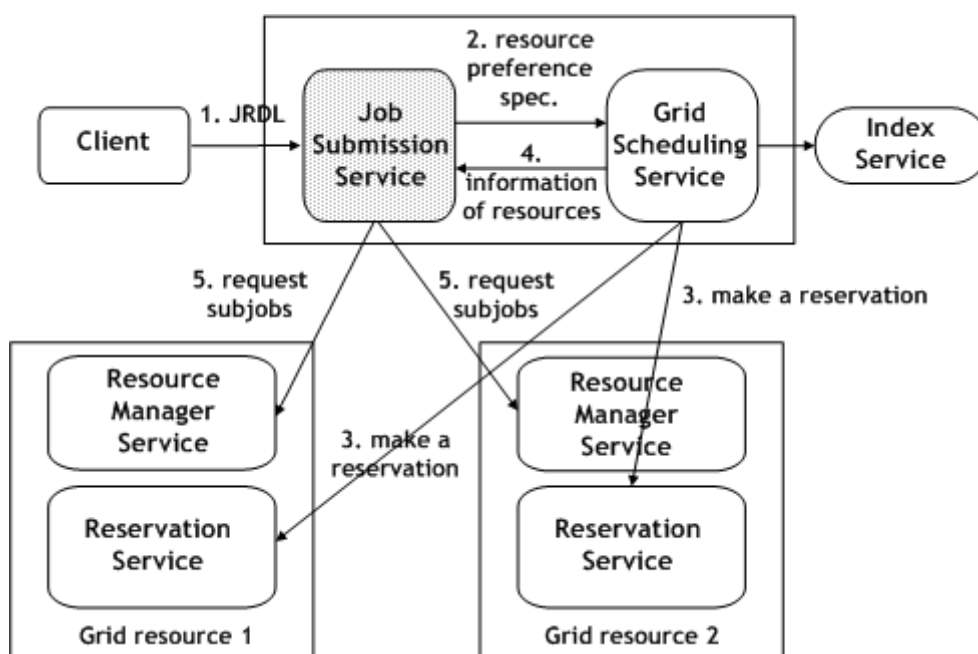


Figure 2. Job Submission Process

### 1.2.3 GSS (Grid Scheduling Service)

#### Key concepts

Grids consist of a large variety of services which reveal accessibilities to resources and the access to a resource is under control of the policies of the resource owners. Besides, complicate bottom layer Grid fabric should be hidden from Grid users. Therefore, the scheduling service which coordinates between various resources and higher level consumers satisfying policies on both sides is acutely needed in the Grid computing environment. The GSS in

GRASP was designed and implemented to do scheduling in such a complex Grid infrastructure for the jobs from various applications.

Major purpose of the GSS is to find resources which meet user's requirements and select resources according to a scheduling algorithm. In order to discover proper resources the GSS queries an information service, GAIS in MoreDream, with resource specification for the job. The GSS does screening process to choose the resources which meet minimal requirements to execute the job.

And then, with the filtered resources, selection is done by the specific scheduling policy. The Grid scheduling service can have several scheduling plugins which implement application-specific policies or scheduling algorithms. The plugin selected by the user will be applied to select most appropriate resources.

Once the selection process is done by a scheduling plugin, the service tries reservations to the selected resources for the time that user have specified in JRDL file. If the reservation fails, the service gains recent information about available resources from the reservation services, does scheduling again, and then retries reservation to the resources. These processes are repeated until the selected resources are confirmed with reservation IDs.

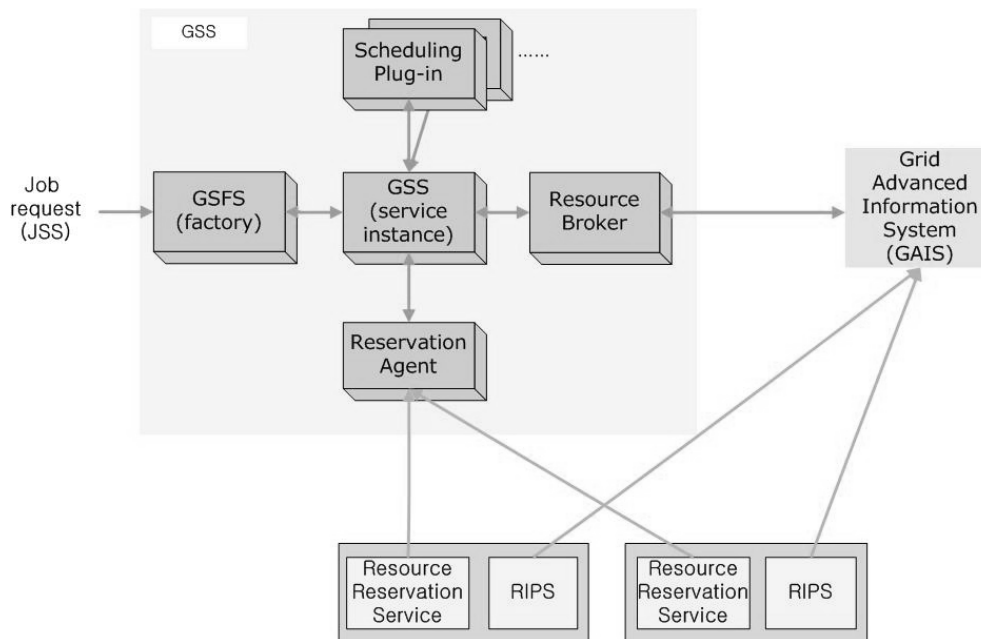
### **Architecture**

The architecture of GSS is described in [Figure 1](#) ! .. Following paragraphs explains each parts of GSS.

The GSS has the factory mechanism i.e. the GSS factory service creates a GSS instance and the created instance deals with the requested job until it gains resources. GSS acquires candidate resources thru Resource Broker. In this step, Resource Broker queries information of available resources to GAIS filtering out the unfit resources.

The scheduling plugin which was specified in the job request takes the job and resource candidates. And then it makes a map between the job and resources according to the scheduling policy. Scheduling plugin can have policies or selection algorithm. GSS has the default plugin, in which a opportunistic load balancing (OLB) algorithm is implemented. OLB assigns each task in the job, in arbitrary order, to the next available node, regardless of the task's expected execution time on that resource. In the distribution of GRASP package, HTC plugin and MPI plugin is included for each type of jobs in addition to the default plugin.

The Reservation Agent takes selected by scheduling plugin and tries to reserve resources for the certainty of the schedule. When the reservation trials are not complete, it asks available node resource capacity to the resource candidates and repeats scheduling and reservation keeping succeeded reservations.



**Figure 3. Architecture of GSS**

## 1.2.4 RMS (Resource Manager Service)

### Key concepts

RMS is a Grid service to enable subjobs to allocate resources and be executed by the local batch scheduler such as PBS. Resources are computational nodes to be managed by the local batch scheduler. The service has the statuses of subjobs which are provided as service data.

### Architecture

Local resource allocation and execution process is illustrated in Figure 4. When RMS receives the execution request for subjobs, the service must allocate local resources. This service can allocate resources when the service gets the permission from RRS, which has established the reservation by the request of Grid scheduling service. RMS then invokes JMS (Job Manager Script) to submit the subjobs to designated local batch scheduler. While the subjobs are running, the service manages the status of each subjob. Because RMS is managed by Job submission service, this service notify the status of subjob to JSS whenever the status changes.

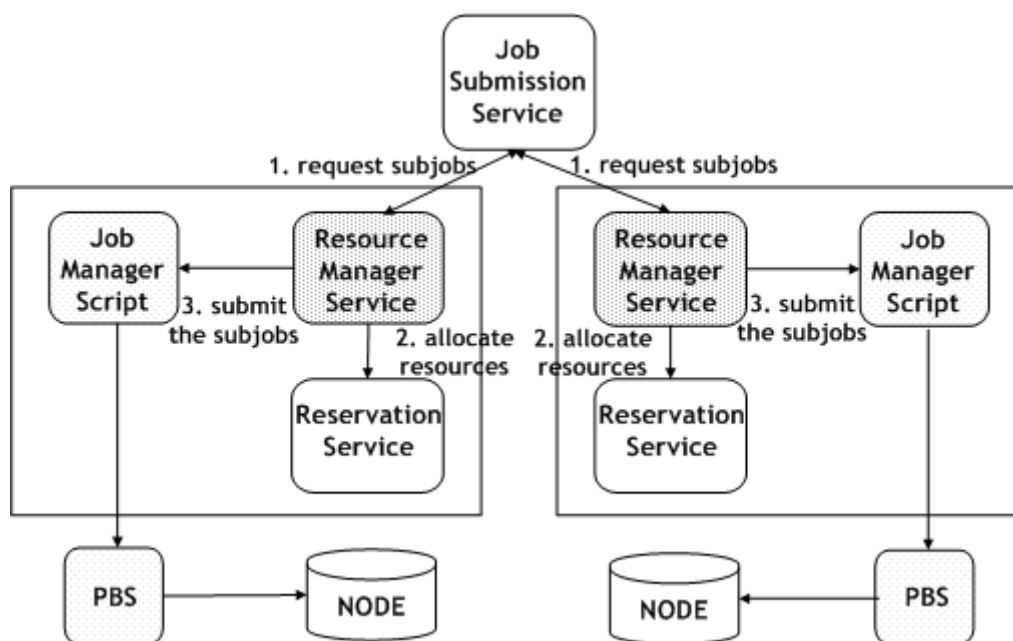


Figure 4. Local Resource Allocation and Execution Process

### JMS (Job Manger Script)

JMS module processes file stage-in, file stage-out, submit, poll, and kill requests instead of RMS. Job manager script is written in PHP language, and can be executed by the java Runtime.exec() method. Requests from the RMS is delivered to JMS as an XML document form. JMS parses the XML and processes the requested function.

JMS is installed under \$GLOBUS\_LOCATION/libexec/grasp-jms-php

JMS can be executed manually in a command line prompt:

```
$ jms -file < filename>
```

The filename is a path name of an XML file, which has the format:

```
<xml>
<action>ACTION</action>
<manager>MANAGER</manager>
<jobtype>JOBTYPE</jobtype>
...
</xml>
```

where,

```
ACTION = proxy_relocate | submit | poll | stage_in | stage_out
MANAGER = pbs | fork
JOBTYPE = single | htc | xmpi
```

File stage-in and stage-out actions use globus-url-copy command to copy files between local file system and remote server. To use PBS manager, OpenPBS must be installed in the local

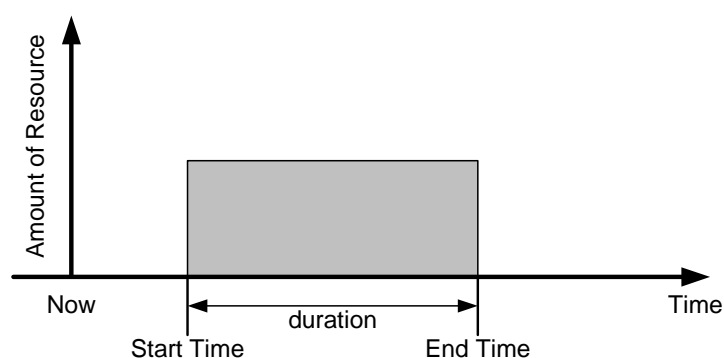
system. The configuration file (config.php) sets up these path information.

### 1.2.5 RRS (Resource Reservation Service)

A Grid resource is composed of many kinds of resources, such as CPUs and memory, storage space, network bandwidth, special purpose instruments. RRS manages reservation of resources which are able to be reserved on the Grid. As a simple case, computing nodes of a cluster system is a kind of a resource which can be reserved. RRS makes a reservation to only computing nodes of a cluster.

In general, to make a reservation to resources, a user should specify the followings:

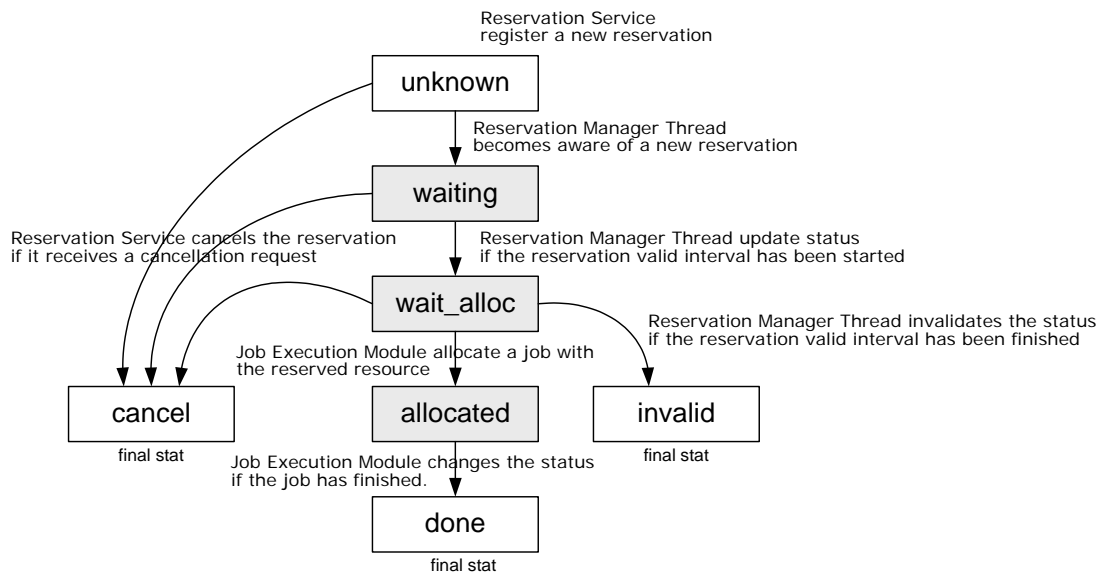
- The start time of reservation
- The end time of reservation (or duration from the start time)
- The kind of resource to reserve
- The identity of reservation maker
- Amount of resource to reserve



**Figure 5. Reservation information: start time, duration, amount of resource, the type of resource and the identity of reservation maker**

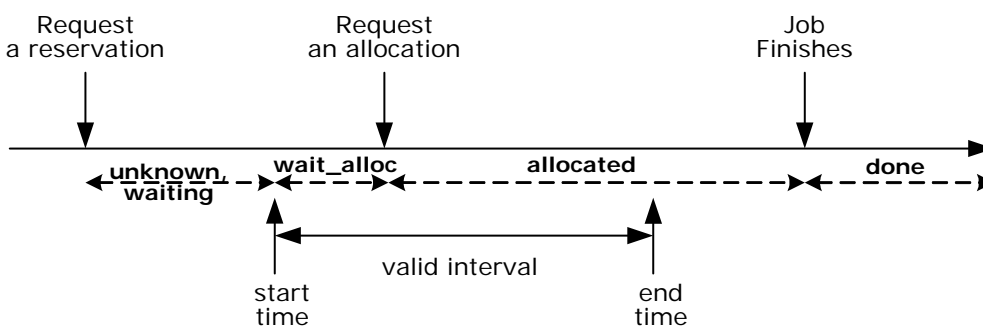
#### Reservation Status

Figure 6 and Figure 7 show the status change of a reservation item. It starts from the 'unknown' status. Until the start time, the reservation is in 'waiting' status, which means that the reservation is valid but it is not ready to be allocated yet. Immediately after the start time the status of the reservation changes to be 'wait\_alloc' status, which means the reservation is valid and it is available for allocation now. After the job runs on the resources, the status of the reservation changes to be 'allocated'. Finally, when the job finished successfully, the status changes to be 'done'. If the valid reservation interval is over with no job allocated, the reservation status becomes to 'invalid' status. Reservations can be canceled if the status is one of 'unknown', 'waiting', or 'wait\_alloc' status.



**Figure 6. Status changes of a reservation**

The 'cancel', 'invalid', and 'done' are final status. The 'waiting', 'wait\_alloc' and 'allocated' status are valid status, which means that reserved resources are not be reserved or available by other users.



**Figure 7. Reservation status changes in a time line**

The Figure 7 depicts the status changes of a reservation in a time line.

## 1.2.6 SRB enabled globus-url-copy

### Overview

globus-url-copy, which is an application of Globus toolkit, copies a file specified by source URL to a location specified by destination URL, using the GASS transfer API. It is used to stage in/out files from/to storage device for executing jobs. All protocols supported by GASS (local file, http, https, ftp, and gsiftp) are supported. Piping to/from stdin/stdout (setting source/dest argument = '-') is also supported. However, it could not retrieve/save data from/to storage not to

be able to use protocol supported by GASS. The Storage Resource Broker (SRB) is client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. SRB support a lot of interfaces for data resources including HRM, HPSS, DB2, Oracle, Illustra, ObjectStore, ADSM, UniTree, UNIX, NTFS, and HTTP. Therefore, we modify `globus-url-copy` application to support SRB protocols as well as GASS protocols for accessing various data resources and replicated data sets.

### SRB URI

The location of data file should be described with URL format to use `globus-url-copy`. Thus, the location of data in SRB could be described with URI format as showed Figure 8. Actually, both `replica` and `resource` are not included in “[http://www1.ietf.org/proceedings\\_new/04nov/IDs/draft-gilbert-srb-uri-00.txt](http://www1.ietf.org/proceedings_new/04nov/IDs/draft-gilbert-srb-uri-00.txt)”. `replica` might be used for accessing replicated data sets. `resource` might be used to specify the resource name for creating new data to SRB.

```

srb:// [username.mdasdomain [.zone] [:password] @] host [:port]
[?replica=replica_id][?resource=resources_name] [/path]

```

where square brackets [...] delineate optional components, the characters `:`, `/`, `@`, and `.` stand for themselves, and spaces should be ignored. If the optional port number is not included, the default port 5544 will be used.

**Figure 8. Syntax for SRB URI**

As mentioned above, the data could be retrieved from SRB server or be saved to SRB server by specified SRB URI format. If there are no attributes except `replica` and `path` in specified format, default configuration will be read in `~/srb/.MdasEnv` file. The `~/srb/.MdasEnv` file includes following default configuration information to connect to SRB server.

- `mdasCollectionName`: default collection name
- `mdasDomainName`: default domain name
- `srbUser`: default SRB user id
- `srbHost`: default host IP of SRB server
- `srbPort`: default host port of SRB server
- `defaultResource`: default storage resource name
- `AUTH_SCHEME`: default authentication mechanism: `PASSWD_AUTH`, `GSI_AUTH`, and `ENCRYPT1`

- `SERVER_DN`: server DN of proxy to invoke SRB server in case of GSI authentication

### SRB Authentication

SRB server accepts ENCRYPT1 or GSI authentication. If `password` exists, it authenticates to SRB server via ENCRYPT1 mechanism. Otherwise, by default, it uses GSI authentication. Because the server DN of proxy to invoke SRB server must be specified in case of GSI authentication, the server DN could be read via `-s`, `-ss`, and `-ds` among options of `globus-url-copy`. If the option is not specified, the server DN should be described by `SERVER_DN` in `~/.srb/.MdasEnv` file.

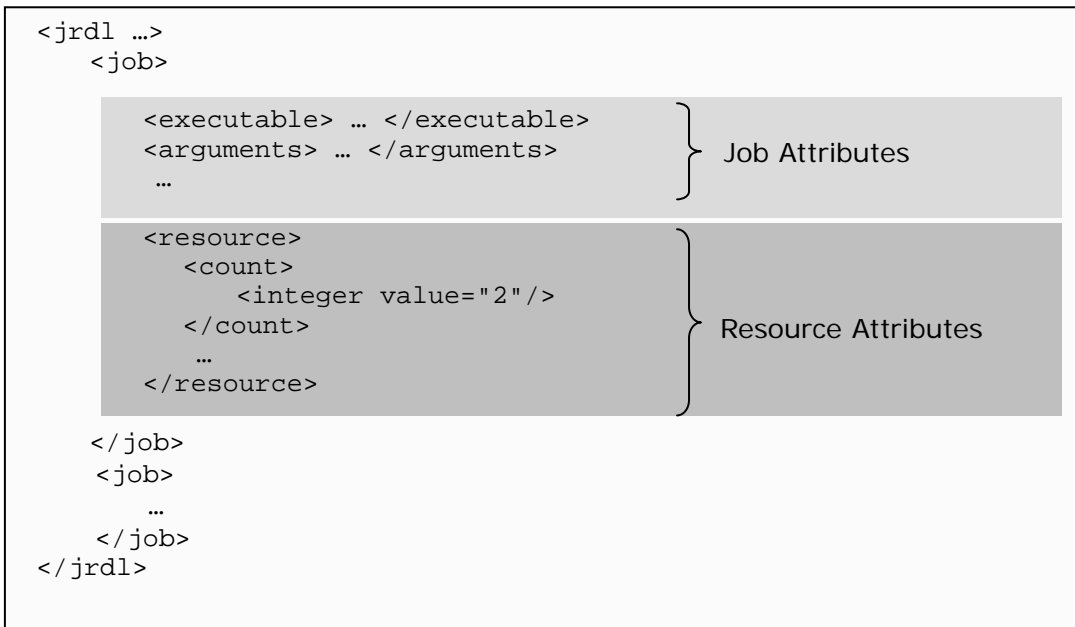
## 1.2.7 JRDL (Job & Resource Description Language)

### Overview

---

The Globus Resource Specification Language (RSL) 2 provides a common interchange language to describe a Grid job. However the RSL 2 does not contain user preferences to select automatically resources for allocating the job. Moreover the RSL 2 has no features to describe elements required to co-allocate the job and to represent sequent jobs. Therefore we provide the Job and Resource Description Language (JRDL) as a general language to describe a job and user preferences required allocating resources for the job in Grid environment based on XML.

JRDL has a collection of jobs, each of which is "an atomic task" of a workflow specification or other kinds of a complex, multi-step application. A job consists of job attributes and resource attributes as showed in Figure 9. Job attributes are elements to need to reserve resources in advance, allocate resources, and execute job. Resource attributes are user preferences to determine resources to execute a job. Here we cover what both kinds of attributes include and how each user should specify each attribute.



**Figure 9. JRDL containing a collection of jobs, each of which has both job and resource attributes**

The job attributes includes executable attributes, job type attribute, file staging attributes, job termination attributes, clean up attributes and co-allocation attributes. Executable attributes are elements for executing a job, including executable, arguments, working directory, environment, standard input, output, and error, and library path. Job type attribute is element to specify the kind of job. File staging attributes are elements for staging in files to execute a job and staging out. Job termination attributes are elements to specify start and termination time of a job to reserve the resources in advance. Resource allocation attributes are related to local resource allocation including local resource address (`resourceManagerContact`) and job manager type (`jobManagerType`) as showed in Figure 10.

```

<jrdl ...>
  <job>

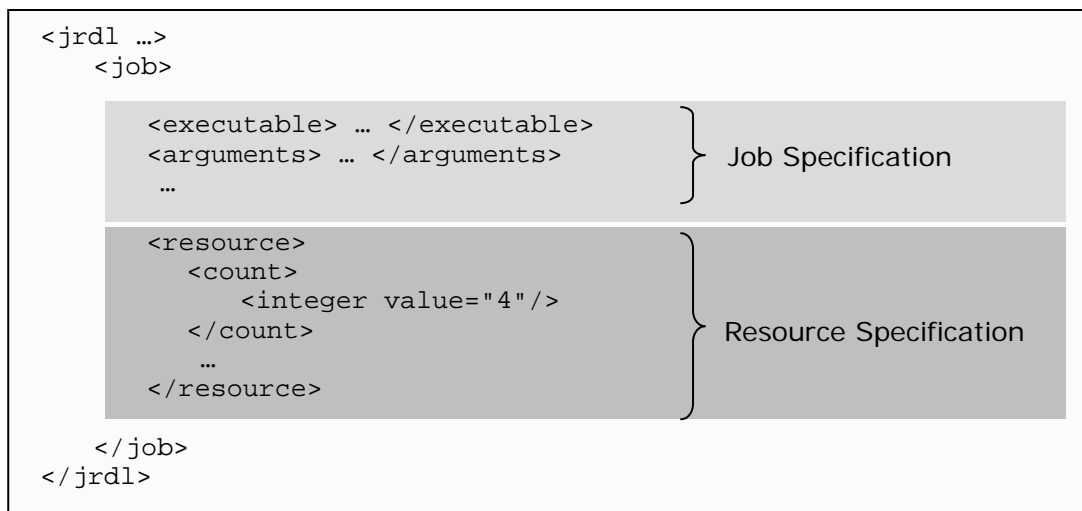
    <executable> ... </executable>
    ...
    <subjob>
      <resourceManagerContact>
        <string>
          <stringElement value=
            "http://eros01.gridcenter.or.kr:8080"/>
        </string>
      </resourceManagerContact>
      <jobManagerType>
        <enumeration>
          <enumerationValue><pbs/></enumerationValue>
        </enumeration>
      </jobManagerType>
      <count>
        <integer value="2"/>
      </count>
      ...
    </subjob>
    <subjob> ... </subjob>

  </job>
</jrdl>

```

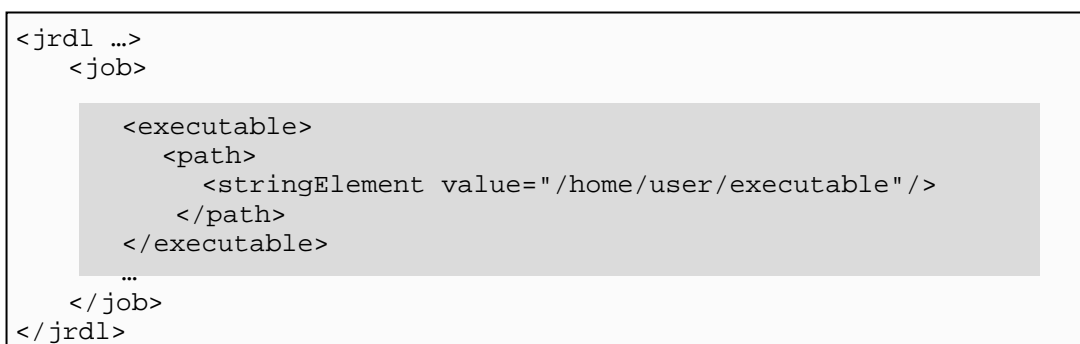
**Figure 10. Job Containing only the Job Specification**

The resource specification is user preference to determine resources to submit a job, including total CPU count, OS type, processor type, size and availability of memory and storage, free CPU, processor load, local job manager type, and scheduler plug-in type. If user specifies the resource allocation attributes in the job specification, which means user want to determine the resources manually without scheduler, he/she does not have to describe the resource specification as showed in Figure 10. However he/she should specify total CPU count if he/she wants to use scheduler as showed in Figure 11.



**Figure 11. Job Containing both the Job Specification and User Preferences to Select Automatically Resources by Scheduler**

Basically, we use RSL2 schema to describe contents of each attribute. For example, job executable attribute must include path element to describe executable file as shown in Figure 12. In addition, as RSL2 provides, each element could include substitutionRef element replaced by the element where substitutionDef element defines. For instance, job executable attribute showed in Figure 12 could be replaced with job executable attribute including substitutionRef element illustrated in Figure 13.



**Figure 12. Job executable attribute**

```

<jrdl ...>
  <substitutionDef name="HOME">
    <stringElement value="/home/user"/>
  </substitutionDef>
  <job>
    <executable >
      <path>
        <substitutionRef name="HOME"/>
        <stringElement value="/executable"/>
      </path>
    </executable >...
  </job>
</jrdl>

```

**Figure 13. Job executable attribute including substitutionRef element**

By default, as illustrated Figure 13, if you want to use substitutionRef element, substitutionDef element must be specified except reserved substitutionDef elements to be defined at local resources as shown in Table 1.

Substitution Definition	Meaning
HOME	Home directory path of user account
LOGNAME	Log name
GLOBUS_LOCATION	Globus location
X509_CERT_DIR	Certificate directory
GLOBUS_HOST_CPUYPE	Host CPU type
GLOBUS_HOST_MANUFACTURER	Host manufacturer
GLOBUS_HOST_OSNAME	Host OS name
GLOBUS_HOST_OSVERSION	Host OS version
GLOBUS_RMS_JOB_CONTACT	The contact address of RMS
GLOBUSRUN_GASS_URL	The address of GASS server that a job submission client invoke automatically

**Table 1. Reserved substitution definition**

## Job attributes

---

Job attributes are elements to need to reserve resources in advance, allocate resources, and execute job, including executable attributes, job type attribute, file staging attributes, job termination attributes, clean up attributes and co-allocation attributes. Executable attributes are elements for executing a job, including executable, arguments, working directory, environment, standard input, output, and error, and library path. Job type attribute is element to specify the kind of job. File staging attributes are elements for staging in files to execute a job and staging out. Job termination attributes are elements to specify start and termination time of a job to reserve the resources in advance. Co-allocation attributes are related to local resource allocation including local resource addresses, job manager type and CPU count.

Basically, a user might want to automatically determine resources by meta-scheduler as illustrated in Figure 9. However If user wants to determine resources manually, co-allocation attributes must be used as showed in Figure 14.

```

<jrdl ...>
  <job>

    <executable> ... </executable>
    ...
    <subjob>
      <resourceManagerContact>
        <string>
          <stringElement value=
            "http://eros01.gridcenter.or.kr:8080"/>
        </string>
      </resourceManagerContact>
      <jobManagerType>
        <enumeration>
          <enumerationValue><pbs/></enumerationValue>
        </enumeration>
      </jobManagerType>
      <count>
        <integer value="2"/>
      </count>
      ...
    </subjob>
    <subjob> ... </subjob>

  </job>
</jrdl>

```

**Co-allocation attributes**

**Figure 14. Co-allocation attributes to specify local resource to submit a job**

When a job is submitted to resources, the job should be divided into several subjobs, each of which might has different job attributes. Therefore, you can specify different job attributes for

each subjob as illustrated in Figure 15. In this job, there are two subjobs, which have different executable file. One is “a.out,” and the other is “b.out”.

```

<jrdl ...>
  <job>
    <executable>
      <path><stringElement value="a.out"/></path>
    </executable>
    <jobType>
      <enumeration>
        <enumerationValue><htc/></enumerationValue>
      </enumeration>
    </jobType>

    <subjob>
      <executable>
        <path><stringElement value="b.out"/></path>
      </executable>
    </subjob>

    <resource>
      <count>
        <integer value="2"/>
      </count>
    </resource>
  </job>
</jrdl>

```

**Figure 15. Job containing different job attributes for a subjob**

## Executable attributes

### Job executable attribute

The source of executable file is local or remote file. If executable file is a form of GASS-compatible URL like “gsiftp://ip/path/file”, executable file could be staged from the GASS-compatible file server.

An element in your document might look like this:

```

<executable>
  <path>
    <stringElement value="EXECUTABLE"/>
  </path>
</executable>

```

### Job arguments attribute

An element in your document might look like this:

```

<arguments>
  <stringArray>
    <string>
      <stringElement value="arg1"/>
    </string>
    <string>
      <stringElement value="arg2"/>
    </string>
  </stringArray>
</arguments>

```

### Job directory attribute

An element in your document might look like this:

```

<directory>
  <path>
    <stringElement value="/path/to"/>
  </path>
</directory>

```

### Environment attribute

An element in your document might look like this:

```

<environment>
  <hashtable>
    <entry name="HOME">
      <stringElement value="/path/to"/>
    </entry>
  </hashtable>
</environment>

```

### Standard input attribute

The source of standard input file is `STDIN`. `STDIN` is local or remote file. If `STDIN` is a form of GASS-compatible URL, `STDIN` could be received from the GASS-compatible file server.

An element in your document might look like this:

```

<stdin>
  <path>
    <stringElement value="STDIN"/>
  </path>
</stdout>

```

### Standard output attribute

The destination of standard output file is `STDOUT`. Standard output file might have multiple

destinations and each destination is local or remote file.

An element in your document might look like this:

```
<stdout>
  <pathArray>
    <path>
      <stringElement value="STDOUT" />
    </path>
  </pathArray>
</stdout>
```

### Standard error attribute

The destination of standard error file is STDERR. Standard error file might have multiple destinations and each destination is local or remote file.

An element in your document might look like this:

```
<stderr>
  <pathArray>
    <path>
      <stringElement value="STDERR" />
    </path>
  </pathArray>
</stderr>
```

### Library path attribute

Job might need to specify library paths. Each library path must be local path.

An element in your document might look like this:

```
<libraryPath>
  <pathArray>
    <path>
      <stringElement value="/librarypath/to"/>
    </path>
  </pathArray>
</libraryPath>
```

## Job type attributes

### Job type attribute

Job type attribute is element to specify the kind of job. JRDL has three kinds of job type: single, xmpi, and htc.

An element in your document might look like this:

```

<jobType>
  <enumeration>
    <enumerationValue>
      <htc/>
    </enumerationValue>
  </enumeration>
</jobType>

```

## File staging attributes

### File staging in attribute

You can specify a list of ("remote URL" "local file") pairs which indicate files to be staged into the cache. Symbolic link from the cache to the "local file" path will be made.

An element in your document might look like this:

```

<fileStageIn>
  <fileInputArray>
    <fileInput>
      <url>
        <urlElement value="gsiftp://ip/path/file"/>
      </url>
      <path>
        <stringElement value="file"/>
      </path>
    </fileInput>
  </fileInputArray>
</fileStageIn>

```

### File staging out attribute

You can specify a list of ("local file" "remote URL") pairs which indicate files to be staged from the job to a GASS-compatible file server.

An element in your document might look like this:

```

<fileStageOut>
  <fileOutputArray>
    <fileOutput>
      <path>
        <stringElement value="file"/>
      </path>
      <url>
        <urlElement value="gsiftp://ip/path/file"/>
      </url>
    </fileOutput>
  </fileOutputArray>
</fileStageOut>

```

## Job termination attributes

These attributes are attributes related to the start time and end time of a job. Job start time and Job start before are mutually exclusive. Therefore, if you want to specify start time of a job, you have to determine between Job start time and Job start before.

## Job start time

If a job should start at a specific time, you should describe a time by Job start time. The dateTime data type is used to specify a date and a time.

The dateTime is specified in the following form "YYYY-MM-DDThh:mm:ss" where:

- YYYY indicates the year
- MM indicates the month
- DD indicates the day
- T indicates the start of the required time section
- hh indicates the hour
- mm indicates the minute
- ss indicates the second

Note: All components are required.

To specify a time zone, you can either enter a dateTime in UTC time by adding a "Z" behind the time.

An element in your document might look like this:

```
<jobStartTime>2004-10-21T01:10:00.000Z</jobStartTime>
```

## Job start before

If a job should start before a specific duration, you should describe a time interval by Job start before. The duration data type is used to specify a time interval.

The time interval is specified in the following form "PnYnMnDTnHnMnS" where:

- P indicates the period (required)
- nY indicates the number of years
- nM indicates the number of months
- nD indicates the number of days
- T indicates the start of a time section (required if you are going to specify hours, minutes, or seconds)
- nH indicates the number of hours
- nM indicates the number of minutes

- nS indicates the number of seconds

An element in your document might look like this:

```
<jobStartBefore>PT10H3M</jobStartBefore>
```

### Job duration

If a job should execute during a specific interval, you should describe a time interval by Job duration. The duration data type is used to specify a time interval.

An element in your document might look like this:

```
<jobDuration>PT10H3M</jobDuration>
```

## Clean up attributes

### Clean up attribute

This attribute specifies files to clean up after a job has completed.

An element in your document might look like this:

```
<fileCleanUp>
  <pathArray>
    <path>
      <stringElement value="cleanUpFile"/>
    </path>
  </pathArray>
</fileCleanUp>
```

## Co-allocation attributes

If you want to manually determine the resources, you should specify co-allocation attributes.

### Resource manager contact attribute

You should specify the contact address of RMS.

An element in your document might look like this:

```
<resourceManagerContact>
  <string>
    <stringElement value="http://service-container"/>
  </string>
</resourceManagerContact>
```

### Job manager attribute

You should specify the type of local job manager. We are supporting two kinds of type: fork and pbs.

An element in your document might look like this:

```
<jobManagerType>
  <enumeration>
    <enumerationValue>
      <pbs/>
    </enumerationValue>
  </enumeration>
</jobManagerType>
```

### Count attribute

You should specify the count of CPU to be allocated at RMS.

An element in your document might look like this:

```
<count>
  <integer value="5" />
</count>
```

## Resource attributes

---

The resource attributes are user preferences to determine resources to submit a job, including total CPU count, OS type, processor type, size and availability of memory and storage, free CPU, processor load, local job manager type, and scheduler plug-in type.

### Count attribute

You must specify the total count of CPU to be allocated to submit a job.

An element in your document might look like this:

```
<count>
  <integer value="5" />
</count>
```

### Operating system attribute

You might need to describe the specific name of operating system of nodes to be selected in resource pools.

An element in your document might look like this:

```

<operatingSystem>
  <name>
    <string>
      <stringElement value="Linux" />
    </string>
  </name>
</operatingSystem>

```

### Processor attribute

You might need to describe the minimum clock speed of processor of nodes to be selected in resource pools. The unit of clock speed is megahertz.

An element in your document might look like this:

```

<processor>
  <clockSpeed>
    <integer value="1000" />
  </clockSpeed>
</processor>

```

### Minimum main memory attributes

#### Minimum RAM available attribute

You might need to describe the minimum available RAM size of nodes to be selected in resource pools. The unit of RAM size is mega byte.

An element in your document might look like this:

```

<minMainMemory>
  <minRAMAvailable>
    <integer value="100" />
  </minRAMAvailable>
</minMainMemory>

```

#### Minimum RAM size attribute

You might need to describe the minimum RAM size of nodes to be selected in resource pools. The unit of RAM size is mega byte.

An element in your document might look like this:

```

<minMainMemory>
  <minRAMSize>
    <integer value="256" />
  </minRAMSize>
</minMainMemory>

```

### Minimum storage attribute

You might need to describe the minimum available storage size of nodes to be selected in resource pools. The unit of storage size is mega byte.

An element in your document might look like this:

```
<minStorage>
  <minStorageAvailable>
    <integer value="1000" />
  </minStorageAvailable>
</minStorage>
```

### Minimum free CPU attribute

You might need to describe the minimum available CPU number of nodes to be selected in resource pools.

An element in your document might look like this:

```
<minFreeCPU>
  <integer value="4" />
</minFreeCPU>
```

### Processor load attributes

#### last15Min attribute

You might need to describe the average processor availability during last 15 minutes of nodes to be selected in resource pools. The average processor availability is expressed as a percentage.

An element in your document might look like this:

```
<processorLoad>
  <last15Min>
    <integer value="10" />
  </last15Min>
</processorLoad>
```

#### last5Min attribute

You might need to describe the average processor availability during last 5 minutes of nodes to be selected in resource pools. The average processor availability is expressed as a percentage.

An element in your document might look like this:

```
<processorLoad>
  <last5Min>
    <integer value="10" />
  </last5Min>
</processorLoad>
```

#### last1Min attribute

You might need to describe the average processor availability during last 1 minute of nodes to

be selected in resource pools. The average processor availability is expressed as a percentage.

An element in your document might look like this:

```
<processorLoad>
  <last1Min>
    <integer value="10" />
  </last1Min>
</processorLoad>
```

### Scheduling type attribute

You might need to describe the type of local job scheduler of nodes to be selected in resource pools. The type of scheduler must be either `fork` or `scheduler`.

An element in your document might look like this:

```
<schedulingType>
  <string>
    <stringElement value="scheduler"/>
  </string>
</schedulingType>
```

### Scheduling plug-in attribute

You might need to describe the name of scheduling plug-in to be selected in resource pools. The name of plug-in must be either `Default` or user defined name. The user defined name could be referenced by querying to the GAIS.

An element in your document might look like this:

```
<schedulingPlugin>
  <string>
    <stringElement value="Default"/>
  </string>
</schedulingPlugin>
```

## JRDL Examples

---

Here are some typical JRDL examples to be used in job submission. There are three kinds of job types in JRDL: `single`, `XMPI`, and `htc`. For each job type, we are giving several examples.

### Single job examples

1. This example is about a single job to use "FORK" local job manager and be submitted to dedicated host "rms\_machine"

```

<?xml version="1.0" encoding="UTF-8"?>
<jrdl xmlns="http://www.moredream.org/namespaces/2003/09/jrdl">
  <job>
    <!-- executable attribute -->
    <executable>
      <path>
        <stringElement value="/bin/echo"/>
      </path>
    </executable>
    <!-- arguments attribute -->
    <arguments>
      <stringArray>
        <string>
          <stringElement value="arg1"/>
        </string>
        <string>
          <stringElement value="arg2"/>
        </string>
      </stringArray>
    </arguments>
    <!-- stdout attribute -->
    <stdout>
      <pathArray>
        <path>
          <stringElement value="stdout"/>
        </path>
      </pathArray>
    </stdout>
    <!-- stderr attribute -->
    <stderr>
      <pathArray>
        <path>
          <stringElement value="stderr"/>
        </path>
      </pathArray>
    </stderr>
    <!-- job type attribute -->
    <jobType>
      <enumeration>
        <enumerationValue>
          <single/>
        </enumerationValue>
      </enumeration>
    </jobType>
    <subjob>
      <resourceManagerContact>
        <string>
          <stringElement value="http://rms_machine"/>
        </string>
      </resourceManagerContact>
      <jobManagerType>
        <enumeration>
          <enumerationValue><b>fork</b></enumerationValue>
        </enumeration>
      </jobManagerType>
    </subjob>
  </job>
</jrdl>

```

```

        </enumeration>
    </jobManagerType>
    <count>
        <integer value="1"/>
    </count>
</subjob>
</job>
</jrdl>

```

2. This example is about a single job to use “PBS” local job manager and be submitted to dedicated host “rms\_machine”

```

<?xml version="1.0" encoding="UTF-8"?>
<jrdl xmlns="http://www.moredream.org/namespaces/2003/09/jrdl">
  <job>
    <!-- executable attribute -->
    <executable>
      <path>
        <stringElement value="/bin/echo"/>
      </path>
    </executable>
    <!-- arguments attribute -->
    <arguments>
      <stringArray>
        <string>
          <stringElement value="arg1"/>
        </string>
        <string>
          <stringElement value="arg2"/>
        </string>
      </stringArray>
    </arguments>
    <!-- stdout attribute -->
    <stdout>
      <pathArray>
        <path>
          <stringElement value="stdout"/>
        </path>
      </pathArray>
    </stdout>
    <!-- stderr attribute -->
    <stderr>
      <pathArray>
        <path>
          <stringElement value="stderr"/>
        </path>
      </pathArray>
    </stderr>
    <!-- job type attribute -->
    <jobType>
      <enumeration>

```

```

        <enumerationValue>
            <single/>
        </enumerationValue>
    </enumeration>
</jobType>
<subjob>
    <resourceManagerContact>
        <string>
            <stringElement value="http://rms_machine"/>
        </string>
    </resourceManagerContact>
    <jobManagerType>
        <enumeration>
            <enumerationValue><b>pbs</b></enumerationValue>
        </enumeration>
    </jobManagerType>
    <count>
        <integer value="1"/>
    </count>
</subjob>
</job>
</jrdl>

```

3. This example is about a single job to allocate resources automatically by meta-scheduler.

```

<?xml version="1.0" encoding="UTF-8"?>
<jrdl xmlns="http://www.moredream.org/namespaces/2003/09/jrdl">
  <job>
    <!-- executable attribute -->
    <executable>
      <path>
        <stringElement value="/bin/echo"/>
      </path>
    </executable>
    <!-- arguments attribute -->
    <arguments>
      <stringArray>
        <string>
          <stringElement value="arg1"/>
        </string>
        <string>
          <stringElement value="arg2"/>
        </string>
      </stringArray>
    </arguments>
    <!-- stdout attribute -->
    <stdout>
      <pathArray>
        <path>
          <stringElement value="stdout"/>
        </path>
      </pathArray>
    </stdout>
  </job>
</jrdl>

```

```

</stdout>
<!-- stderr attribute -->
<stderr>
  <pathArray>
    <path>
      <stringElement value="stderr"/>
    </path>
  </pathArray>
</stderr>
<!-- job type attribute -->
<jobType>
  <enumeration>
    <enumerationValue>
      <single/>
    </enumerationValue>
  </enumeration>
</jobType>
<resource>
  <count>
    <integer value="1"/>
  </count>
</resource>
</job>
</jrnl>

```

## 1.2.8 Client Tools

GRASP provides client tools. They provide three user interfaces: a command line interface (CLI), a graphic user interface (GUI), and web interface (WI). They have same functionality for creating and modifying JRDL for a job, submitting the job to JSS, and controlling and monitoring the job.

### Command Line Interface (CLI): *grasprun*

The CLI lets you execute commands via *grasprun* at the shell prompt. The CLI could be run at Linux or Windows environment. The user should install the library and certificates to use *grasprun* and know the syntax of JRDL to describe a job. Detail usage of the CLI is described in the appendix of this document.

### Graphic User Interface (GUI)

The GUI is a graphic interface based on Java SWING, which is OS independent. While the user must write a JRDL manually in case of *grasprun*, GUI provides convenient interface to load and write the JRDL.

**Web Interface (WI): Job Submission Portlet**

The WI is a web interface based on Gridsphere, which provides an open-source portlet based Web portal. While the user should install the library and certificates in case of both CLI and GUI, he/she does not have to care about installation as well as the syntax of JRDL in web interface. We have implemented Job submission portlet enabling the user to easily make a JRDL, load the JRDL, submit a job to JSS, and monitor the submitted job.

## 1.3 Getting help

If you have questions about GRASP or have found a bug, please send an email to [kmi@moredream.org](mailto:kmi@moredream.org). For up-to-date information about GRASP, please visit the web site <http://kmi.gridcenter.or.kr/>.

## 2 Installation and Configuration

### 2.1 Support software

#### 2.1.1 Required

- OS: Linux (RedHat 7.3 or more are recommended)
- Globus Toolkit 3 (developed under 3.2.1)

### 2.2 Installing support softwares

#### 2.2.1 Installing Globus Toolkit

1. Download all source code from <http://www.globus.org>
2. As globus, untar the source installer.
3. Make sure that ANT\_HOME and JAVA\_HOME are set, and that ant and java are on your PATH.
4. Run
 

```
# ./install-gt3 /path/to/install
```
7. Configure the Globus Toolkit 3.2, looking through [http://www-unix.globus.org/toolkit/docs/3.2/installation/install\\_config.html](http://www-unix.globus.org/toolkit/docs/3.2/installation/install_config.html)

### 2.3 Installing GRASP Client

We provide two kinds of packages: one is a package containing both a command line interface (CLI) and a graphic user interface (GUI), and the other is a package providing a web interface (WI) based on Gridsphere portlet.

#### Download

<http://kmi.moredream.org/downloads/index.php>

You can download the whole package of GRASP from the web site written above. And then you can get the client tools from following files of each component of GRASP.

```
Grasp-0.9
|-- jobsubmissioin-0.1.tar.gz
|-- mrmfs-0.11.tar.gz
|-- gridscheduling-0.1-src.tar.gz
|-- globus_gass_copy-srb-0.1.tar.gz
```

```
|-- grasp-client-0.1.tar.gz
|-- jobsubmission-portlet-0.1.tar.gz
```

## 2.3.1 Client Tools

### A client package containing both the CLI and the GUI: grasp-client-0.1.tar.gz

This package could be run on both Windows and Linux. This package is also included in JSS package.

#### Required

- Globus Toolkit 3.2.1 WS Core (<http://www-unix.globus.org/toolkit/downloads/3.2.1/#core>)

#### Download

<http://kmi.moredream.org/downloads/index.php>

#### Linux Installation

Note: Before installing, you must set environment variable GLOBUS\_LOCATION and copy grasp-client-0.1.tar.gz to \$GLOBUS\_LOCATION.

```
$ cd $GLOBUS_LOCATION
$ tar zxvf grasp-client-0.1.tar.gz
```

#### Linux Configuration

You can specify default factory address in configuration file: \$HOME/.globus/.grasprun.  
factory=http://ipaddress:port

#### Windows Installation

Note: Before installing, you should set environment GLOBUS\_LOCATION

1. unzip grasp-client-0.1.zip to %GLOBUS\_LOCATION%

#### Windows Configuration

You can specify default factory address in configuration file: %HOME%\globus\grasprun, where %HOME% is home directory. In case of Windows XP and 2000, home directory is "C:\Documents and Settings\username\  
factory=http://ipaddress:port

### A Job submission portlet package providing the WI: jobsubmission-

**portlet-0.1.tar.gz****Required**

- Gridsphere 2.0.1
- Gridportlets portlet

**Download**

<http://kmi.moredream.org/downloads/index.php>

**Installation**

```
$ cd $GRIDSPHERE_LOCATION/projects
$ tar zxvf jobsubmission-portlet-0.1.tar.gz
$ cd jobsubmission-portlet
$ ant install
```

### 3 References

<http://www-unix.globus.org/toolkit/docs/3.2/installation/index.html>

<http://java.sun.com/j2se/1.4.2/index.jsp>

<http://ant.apache.org/manual/index.html>

[http://testbed.gridcenter.or.kr/software/OpenPBS/doc/v2.3\\_admin.pdf](http://testbed.gridcenter.or.kr/software/OpenPBS/doc/v2.3_admin.pdf)

<http://www.gridisphere.org/gridsphere/docs/index.html>

## Appendix A: A Client Program Providing Command Line Interface (CLI) : grasprun

grasprun is a client program providing CLI to submit, monitor, and control a job as well as validate a job description language called JRDL. Here we describe the usage of grasprun for each functionality.

### A.1 Validate a JRDL

grasprun can parse a job description language called JRDL, validate parsed JRDL, and then show each attribute of JRDL. You might specify like this:

```
$ grasprun -parse -f cpi_spec.xml
Job Type: XMPI
Executable: /home/guest01/kmi-test/cpi/cpi[local]
Environment: LD_LIBRARY_PATH[/usr/local/gt3.2.1/lib]
Stdout: stdout
Stderr: stderr
Subjob [0]
  Resource Manager Contact: http://vega01.gridcenter.or.kr:8080
  Job Manager Type: pbs
  Count: 10
$
```

, where `-p` or `-parse` option is parsing operation, and `-f` or `-file` option is JRDL file name.

### A.2 Job Submission

grasprun provides two kinds of job submission modes: batch and interactive mode. If you want to submit, monitor, and control a job interactively, you might specify like this:

```

$ grasprun -f a.xml -factory http://factory_address:8080 -o
Job ID: http://
factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/h
ash-23449824-1112770816197
WAITING FOR JOB TO FINISH
===== Job Status =====
Job Status: Pending
[vega01.gridcenter.or.kr, PBS] Unsubmitted
=====
===== Job Status =====
Job Status: Active
[vega01.gridcenter.or.kr, PBS] Active(vega03+vega02)
=====
pi is approximately 3.1416009869231249, Error is 0.0000083333333318
wall clock time = 4.702783
Process 0 on vega03.gridcenter.or.kr
===== Job Status =====
Job Status: Done
[vega01.gridcenter.or.kr, PBS] Done(vega03+vega02)
=====
It takes 87.0 seconds.
$

```

, where `-f` or `-file` option is JRDL file name, `-factory` option is a service container address including JSS service, and `-o` option let standard output of job to print the console. If you configure the default setting for grasprun in `~/.globus/grasprun` file as following, you do not have to specify `-factory` option. Additionally, the message of job status is fully shown, as property "full" set to "true".

```

factory=http://factory_address:8080
full=true

```

If you want to submit a job in a batch mode, you might specify like this:

```

$ grasprun -file cpi_spec.xml -b
Job ID:
http://factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/hash-28099439-1112773501104
$

```

## A.3 Job List and Information

grasprun can list the submitted job. Especially for batch job, it is essential to get the list. You might specify like this:

```

$ grasprun -list
Job ID:
http://factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/hash-23449824-1112770816197
Job ID: http://
factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/hash-28099439-1112773501104
$

```

, where `-list` or `-l` option is list operation.

And also information of the job could be checked like this:

```

$                               grasprun                               -info
http://factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/hash-23449824-1112770816197
Job                               ID:
http://150.183.234.231:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/hash-23449824-1112770816197
Submitted Time: Thu Apr 07 09:40:41 KST 2005
Job Type: XMPI
Executable: /home/guest01/kmi-test/cpi/cpi[local]
Environment: LD_LIBRARY_PATH[/usr/local/gt3.2.1/lib]
Stdout: stdout, /dev/stdout[local]
Stderr: stderr, /dev/stderr[local]
Subjob [0]
    Resource Manager Contact: http://vega01.gridcenter.or.kr:8080
    Job Manager Type: pbs
    Count: 10
$

```

, where `-info` or `-i` option is job information operation.

## A.4 Job Monitoring and Controlling

Submitted job might be checked the status of the job like this:

```

$                               grasprun                               -status
http://factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactoryService/hash-23449824-1112774942696
===== Job Status =====
Job Status: Done
[eros01.gridcenter.or.kr, FORK] Done
=====
$

```

, where `-status` or `-s` option is status operation.

And also, submitted job might be killed like this:

```
$          grasprun          -kill
http://factory_address:8080/ogsa/services/base/grasp/JobSubmissionFactorySe
rvice/hash-23449824-1112774942696
$
```

, where `-kill` or `-k` option is kill operation.