NORDUGRID-TECH-18
3/9/2009

ARC BATCH SYSTEM BACK-END INTERFACE GUIDE WITH SUPPORT FOR GLUE 2

Description and developer's guide for ARC1

Adrian Taga*, Thomas Frågåt†

*v.a.taga@fys.uio.no
†thomas.fragat@fys.uio.no

Contents

1 Introduction **4**
- 1.1 ARC Classic . . . 4
- 1.2 ARC1 . . . 4

2 Job control interface **4**
- 2.1 Submit-LRMS-job . . . 4
- 2.2 Cancel-LRMS-job . . . 5
- 2.3 Scan-LRMS-job . . . 5
- 2.4 Configuration parser . . . 5
  - 2.4.1 Functions . . . 5

3 Information System interface **6**
- 3.1 The LRMS Perl module interface . . . 6
  - 3.1.1 get_lrms_options_schema . . . 6
  - 3.1.2 lrms_info . . . 6

4 Batch system specifics **8**
- 4.1 PBS . . . 8
  - 4.1.1 Recommended batch system configuration . . . 8
  - 4.1.2 Relevant `arc.conf` options . . . 9
  - 4.1.3 Implementation details . . . 10
  - 4.1.4 Known limitations . . . 10
- 4.2 Fork . . . 10
  - 4.2.1 Recommended batch system configuration . . . 10
  - 4.2.2 Relevant arc.conf options . . . 10
  - 4.2.3 Implementation details and known limitations . . . 11
- 4.3 SGE . . . 11
  - 4.3.1 Recommended batch system configuration . . . 11
  - 4.3.2 Relevant arc.conf options . . . 11
  - 4.3.3 Implementation details . . . 11
  - 4.3.4 Known limitations . . . 12

5 The ARC GLUE2 implementation **12**
- 5.1 ComputingService . . . 13
- 5.2 ComputingEndpoint . . . 13
- 5.3 ComputingShare . . . 13
- 5.4 ExecutionEnvironment . . . 15
- 5.5 ApplicationEnvironment . . . 15
- 5.6 ComputingActivity . . . 15
- 5.7 MappingPolicy and AccessPolicy . . . 15

1 Introduction

This document describes the next generation Advanced Resource Connector [**?**] (ARC) batch system back-end infrastructure. It aims to describe the infrastructure in enough detail that a developer can add a new local resource management system (LRMS) to the ARC middleware. It also intends to serve as a reference manual describing existing batch system interfaces. Note that certain details of the interface which are not yet finalized are not described, but will be included in a future revision of this document.

The batch system back-ends are what tie the ARC grid middleware (through the ARC Resource-coupled EXecution service, A-REX [**?**]) to the underlying cluster management system or LRMS. The back-ends consist of set of a shell and Perl scripts whose role are twofold:

1. to allow the GridManager (GM), which is resided in A-REX, to control jobs in the LRMS including job submit, cancel operations etc.
2. to collect information about jobs, users, the batch system and the cluster itself for the Information System.

The former will be referred to as the job control back-end interface while the latter is the information system interface of the batch system back-ends. These two will be treated separately in the following sections.

1.1 ARC Classic

The ARC Classic has its own solution, please refer to [**?**].

1.2 ARC1

As of ARC version 0.9, which is to be the next generation ARC, the scripts are located in different directories within the NorduGrid subversion tree [**?**]. The job control interface scripts can be found under

```
arc1/trunk/src/services/a-rex/lrms/
```

while the information collectors are located in

```
arc1/trunk/src/services/a-rex/infoproviders/.
```

The backend scripts install along with the ARC1 code.

The next generation of ARC supports both the classic NorduGrid information schema [**?**] and currently a minimal set of the GLUE specification version 2.0 schema [**?**].

2 Job control interface

The job control part of the LRMS interface is handled by the Grid Manager [**?**]. It takes care of preparing a native batch system submission script, managing the actual submission of the batch system job, cancellation of job on request and scanning for completed batch jobs. Besides the LRMS job control interface it is also the GM which provides e.g. the data staging and communication with the grid client, provides RTE environments, arranges file staging (to the node via LRMS capability), dealing with stdout/stderr, etc. The job control batch system interface of the GM requires three programs. These programs can be implemented any way the designer sees it fits, but all the existing back-end interfaces use shell scripting for portability and ease of tailoring to a specific site. The GM will call the following programs: cancel-LRMS-job, submit-LRMS-job, and scan-LRMS-job where LRMS is replaced with the short hand name for the LRMS; e.g. cancel-pbs-job. The scripts are described one by one in the following subsections. Useful information can also be found in the Section "8.6 LRMS Support" and Section "8.7 Runtime Environment" of the Grid-Manager guide [**?**].

2.1 Submit-LRMS-job

The submit program is the most involved. It is called by the GM once a new job arrives and needs to be submitted to the LRMS. It is given the GRAMi file as argument on execution. The GRAMi file is a file in the job control directory containing the job description in a flat list of key-value pairs. This file is created by GM and is based on the JSDL job description. Submit-LRMS-job then has to set up the session directories, run-time environment and anything else needed. Then it submits the job to the local LRMS. This is normally done by generating a native job script for the LRMS and then running the local submit command, but it can also be done through an API if the LRMS supports it.

2.2 Cancel-LRMS-job

If a grid user cancels his job, the message will reach the grid-manager. The manager will then call the cancel-LRMS-job for the suitable back-end. The cancel script is called with the GRAMi file containing information about the job such as the job id in the LRMS. Cancel-LRMS-job must then use that information to find the job and remove it from the queue or actually cancel it if it is running in the LRMS.

2.3 Scan-LRMS-job

The scan-LRMS-job is run periodically. Its job is to scan the LRMS for jobs that have finished. Once it has found a finished job it will write the exit-code of that job to the file job.{gridid}.lrms_done in the ARC job status directory[‡]. Then it will call the gm-kick program to notify GM about the finished job. Subsequently, the GM starts finalizing the job.

Generally, two approaches are taken to find jobs which are finished in LRMS. One is to directly ask the LRMS. Since all started grid jobs have its own status file[§] found in the job status directory, this can be done by checking if the status is "INLRMS" in this file. If so, a call to the LRMS is made asking for the status of the job (or jobs if several jobs have status "INLRMS"). If it is finished, it is marked as such in the job status directory, and the gm-kick program is activated. For most LRMSs the information about finished jobs are only available for a short period of time after the job finished. Therefore appropriate steps have to be taken if the job has the status "INLRMS" in the job status directory, but is no longer present in the LRMS. The normal approach is to analyze the job's status output in the session directory.

The second approach is to parse the LRMSs log files. This method has some drawbacks like e.g.: the GM has to be allowed read access to the logs. The back-end will then have to remember where in the log it was last time it ran. This information will have to be stored in a file somewhere on the front-end.

2.4 Configuration parser

Some of the back-ends will need information from the configuration file. Since this functionality can be shared among the back-ends, a configuration file parser written in bash has been provided separately for easy maintenance and extendability.

2.4.1 Functions

`config_parse_file <config_file>`

> Parses a config file. It returns exit status 1 if the file cannot be read, or 0 otherwise. Badly formed lines are silently ignored. Option values can be surrounded by single quotes or double quotes. Values without quotes are also accepted. Currently, multi-valued options are not supported. Only the last defined value is retained.

`config_import_section <section_name>`

> Imports options from section <section_name> of the config file into environment variables of the form 'CONFIG_optionname'. Already existing environment variables are overwritten.

Example:

```
source $ARC_LOCATION/libexec/config_parser.sh
config_parse_file /etc/arc.conf || exit 1
config_import_section common
config_import_section grid-manager
config_import_section infosys
echo $CONFIG_pbs_bin_path
```

[‡] normally /var/spool/nordugrid/jobstatus/, but can be set via the controldir variable of arc.conf

[§] job.{gridid}.status

3 Information System interface

The main purpose of the information system batch interface is to populate the NorduGrid information model and the GLUE2 model with locally collected information obtained from the batch system. It is important to recall that the locally collected information is broader than what the batch-system can offer, information taken from the grid-layer (mostly A-REX), from the front-end machine, and from the `arc.conf` configuration file are also needed and used to populate the NorduGrid information model [**?**] and the GLUE2 schema [**?**].

The information system interface consists of a set of Perl scripts which generates an XML output to `STDOUT` by one invocation of the Perl script named `CEinfo.pl`. The XML output includes two representations of the output data, namely the classic NorduGrid information schema and an incomplete GLUE2 schema representation.

The information collector scripts are divided between several separate scripts:

- `CEinfo.pl` - driver for information collection. It calls all other information collectors and prints the results in XML. The information collection is done by one single invocation of this script.
- `InfoCollector.pm` - base class for all information collectors (i.e.: all files `*Info.pm`).
- `InfoChecker.pm` - used by `InfoCollector` to validate options and results against a "schema".
- `GMJobsInfo.pm` - collects information about jobs from GM status files.
- `HostInfo.pm` - collects other information that can be collected on the front end (hostname, software version, disk space for users, installed certificates, Runtime environments, etc.)
- `LRMSInfo.pm` - collects information that is LRMS specific (queues, jobs, local user limits, etc.) by calling the appropriate LRMS plugin. It also does validation of input options to the plugin and of the data returned by the plugin.
- `<BATCH_SYSTEM_NAME>.pm` - plugins for LRMSInfo. Only Fork, SGE and PBS are updated to the new framework.
- `ARC0ClusterInfo.pm` - combines all information about A-REX and produces information structured according to the classic NorduGrid schema.
- `ARC1ClusterInfo.pm` - combines all information about A-REX and produces information structured according to the GLUE2 schema.

The following subsection describes the LRMS interface part of the information system. To support a particular LRMS a dedicated Perl module should be written that implements the interface presented in the next subsections. Furthermore, hooks should be added to the `<BATCH_SYSTEM_NAME>.pm` module, so that it uses the correct Perl module depending on the LRMS type.

3.1 The LRMS Perl module interface

Each LRMS module should implement two functions: `lrms_info` and `get_lrms_options_schema`.

3.1.1 get_lrms_options_schema

This function is called without arguments and should return a "schema" hash declaring the configuration options specific for this LRMS plugin. This "schema" conforms to the format understood by `InfoChecker.pm` and is used to validate input options to the plugin.

3.1.2 lrms_info

The `lrms_info` function returns all LRMS specific information. This includes information about the cluster, queues, jobs and local user limits. The function is called with a hash of hashes containing all options required by the LRMS plugin. Some options are common to all plugins, and are described in table 1.

The `lrms_info` function should then return a hash of hashes with the structure described in table 3.

Table 1: The `lrms_options` hash used as input for the `lrms_info` function

key	value
jobs	array with local job IDs
queues	hash with queue names as keys and values being `queue_options` hashes (table 2)

Table 2: The `queue_options` hash

key	value
users	array with local UNIX user names

Table 3: The `lrms_info` hash returned by the `lrms_info` function

key	value
lrms_type	the type of LRMS e.g PBS
lrms_version	the version of the LRMS
totalcpus	total number of CPUs in the cluster
queuedcpus	total number of CPUs requested by LRMS queuing jobs (both grid and non-grid)
usedcpus	CPUs in the LRMS that are currently in use either by grid or non-grid jobs
cpudistribution	number of CPUs in a node and number of each type e.g. "8cpu:5 2cpu:100"
queues	a hash with keys being queue names and values being `queue_info` hashes as described in table 4
jobs	a hash with keys being LRMS job IDs and values being `jobs_info` hashes as described in table 6

Table 4: The `queue_info` hash

key	value
status	available slots in queue, negative number signals error
maxrunning	limit on number of running jobs
maxqueuable	limit on number of jobs queued on this queue
maxuserrun	limit on number of running jobs per user
maxcputime	limit on maximum CPU time for a job in this queue in minutes
mincputime	limit on minimum CPU time for a job in this queue in minutes
efaultcput	default CPU time limit for a job in this queue in minutes
maxwalltime	limit on maximum wall time[a] for a job in this queue in minutes
minwalltime	limit on minimum wall time for a job in this queue in minutes
defaultwallt	default wall time limit for a job in this queue in minutes
running	number of CPUs used by running jobs (both grid and non-grid) in the queue
queued	number of CPUs requested by queuing jobs in the queue
totalcpus	number of CPUs available to the queue
users	a hash with keys being local UNIX user names and values being `users_info` hashes as described in table 5

[a] Also known as Wall clock time, `http://en.wikipedia.org/wiki/Wall_clock_time`

Table 5: The `users_info` hash

key	value
freecpus	number of freely available CPUs with their time limits in minutes (see ComputingService.FreeSlotsWithDuration in [**?**])
queuelength	estimated queue length for the specified user

Table 6: The `jobs_info` hash

key	value
status	LRMS jobstatus mapping[a]: Running ⇒ 'R', Queued ⇒ 'Q', Suspended ⇒ 'S', Exiting ⇒ 'E', Other ⇒ 'O'
rank	the job's position in the LRMS queue
mem	the memory usage of the job in kBs
walltime	consumed wall time in minutes
cputime	consumed CPU-time in minutes
reqwalltime	wall time requested by job in minutes
reqcputime	CPU-time requested by job in minutes
nodes	list of execution hosts
comment	array of string comments about the job in LRMS, can be an empty array

[a] LRMS job states mapping is described in[**?**]

4 Batch system specifics

This section presents the batch system specific implementation details including information on supported versions, constraints on batch system configuration, known limitations, `arc.conf` parameters, and a list of batch system features being utilized within the interface are described.

4.1 PBS

The Portable Batch System (PBS) is one of the most popular batch systems. PBS comes in many flavours such as OpenPBS (unsupported), Terascale Open-Source Resource and QUEue Manager (TORQUE) and PBSPro (currently owned by Altair Engineering). ARC supports all the flavours and versions of PBS.

4.1.1 Recommended batch system configuration

PBS is a very powerful LRMS with dozens of configurable options. Server, queue and node attributes can be used to configure the cluster's behaviour. In order to correctly interface PBS to ARC (mainly the information provider scripts) there are a couple of configuration REQUIREMENTS asked to be implemented by the local system administrator:

1. The computing nodes MUST be declared as cluster nodes (job-exclusive), at the moment time-shared nodes are not supported by the ARC setup. If you intend to run more than one job on a single processor then you can use the virtual processor feature of PBS.

2. For each queue, you MUST set one of the max_user_run or max_running attributes and its value SHOULD BE IN AGREEMENT with the number of available resources (i.e. don't set the max_running = 10 if you have only six (virtual) processors in your system). If you set both max_running and max_user_run then obviously max_user_run has to be less or equal to max_running.

3. For the time being, do NOT set server limits like max_running, please use queue-based limits instead.

4. Avoid using the max_load and the ideal_load directives. The Node Manager (MOM) configuration file (<PBS home on the node>/mom_priv/config) should not contain any max_load or ideal_load directives. PBS closes down a node (no jobs are allocated to it) when the load on the node reaches the max_load value. The max_load value is meant for controlling time-shared nodes. In case of job-exclusive nodes there is no need for setting these directives, moreover incorrectly set values can close down your node.

5. Routing queues are not supported, those cannot be used within ARC.

Additional useful configuration hints:

- If possible, please use queue-based attributes instead of server level ones (for the time being, do not use server level attributes at all).
- You may use the "acl_user_enable = True" with "acl_users = user1,user2" attribute to enable user access control for the queue.
- It is advisory to set the max_queuable attribute in order to avoid a painfully long dead queue.
- You can use node properties from the `<PBS home on the server>/server_priv/nodes` file together with the `resources_default.neednodes` to assign a queue to a certain type of node.

Checking your PBS configuration:

- The node definition can be checked by `<PBS installation path>/bin/pbsnodes -a`. All the nodes MUST have ntype=cluster.
- The required queue attributes can be checked as `<PBS installation path>/bin/qstat -f -Q queuename`. There MUST be a max_user_run or a max_running attribute listed with a REASONABLE value.

4.1.2 Relevant `arc.conf` options

Below the PBS specific configuration variables are collected.

- The PBS batch system back-end is enabled via setting the `lrms="pbs"` in the `[common]` configuration block. No need to specify the flavour or the version number of the PBS, simply use the "`pbs`" keyword as LRMS configuration value.
- `pbs_bin_path` configuration variable of the `[common]` block should be set to the path to the qstat,pbsnodes,qmgr etc PBS binaries.
- `pbs_log_path` configuration variable of the `[common]` block should be set to the path of the PBS server logfiles which are used by the GM to determine whether a PBS job is completed. If not specified, the GM will use the `qstat` command to find completed jobs.
- `lrmsconfig` from the `[cluster]` block can be used as an optional free text field to describe further details about the PBS configuration (e.g. `lrmsconfig="single job per processor"`).
- `dedicated_node_string` from the `[cluster]` block specifies the string which is used in the PBS node config to distinguish the grid nodes from the rest. Suppose only a subset of nodes are available for grid jobs, and these nodes have a common `node property` string, this case the `dedicated_node_string` should be set to this value and only the nodes with the corresponding PBS `node property` are counted as grid enabled nodes. Setting the `dedicated_node_string` to the value of the PBS `node property` of the grid-enabled nodes will influence how the totalcpus, user freecpus is calculated. No need to set this attribute if the cluster is fully available for the grid and the PBS configuration does not use the `node property` method to assign certain nodes to grid queues.
- `[queue/queuename]` block. For each grid-enabled (or grid visible) PBS queue a corresponding `[queue]` block must be defined. `queuename` should be the PBS queue name.
- `scheduling_policy` from the `[queue/queuename]` block describes the scheduling policy of the queue. PBS by default offers the FIFO scheduler, many sites run the MAUI. At the moment `FIFO` & `MAUI` are supported values. If you have a MAUI scheduler you should specify the "MAUI" value since it modifies the way the queue resources are calculated. By default the "FIFO" scheduler type is assumed.

- maui_bin_path from the [queue/queuename] block sets the path of the MAUI commands like showbf when "MAUI" is specified as scheduling_policy value. This parameter can be set in the [common] block as well.
- queue_node_string of the [queue/queuename] block can be used similar to the dedicated_node_string. In PBS you can assign nodes to a queue (or a queue to nodes) by using the node property PBS node configuration method and assigning the marked nodes to the queue (setting the resources_default.neednodes = queue_node_string for that queue). This parameter should contain the node property string of the queue-assigned nodes. Setting the queue_node_string changes how the queue-totalcpus, user freecpus are determined for this queue.

4.1.3 Implementation details

The job control batch interface makes use of the qsub command to submit native PBS job scripts to the batch system. The following options are used:

-l nodes, cput, walltime, pvmem, pmem,

-W stagein, stageout

-e, -j eo

-q

-A

-N

For job cancellation the qdel command is used. To find completed jobs, i.e. to scan for finished jobs the qstat command or the PBS server log file is used.

The information system interface utilizes the qstat -f -Q queuename and qstat -f queuename commands to obtain detailed job and queue information. qmgr -c "list server" is used to determine PBS flavour and version. The pbsnodes command is used to calculate total/used/free cpus within the cluster. In case of a MAUI scheduler the showbf command is used to determine user freecpu values. All these external PBS commands are interfaced via parsing the commands' output.

4.1.4 Known limitations

Some of the limitations are already mentioned under the PBS deployment requirements. No support for routing queues, difficulty of treating overlapping queues, the complexity of node string specifications for parallel jobs are the main shortcomings.

4.2 Fork

The Fork back-end is a simple back-end that interfaces to the local machine, i.e.: there is no batch system underneath. It simply forks the job, hence the name. The back-end then uses standard posix commands (e.g. ps or kill) to manage the job.

4.2.1 Recommended batch system configuration

Since fork is a simple back-end and does not use any batch system, there is no specific configuration needed for the underlying system.

4.2.2 Relevant arc.conf options

- The Fork back-end is enabled by setting lrms="fork" in the [common] configuration block.
- The queue must be named “fork” in the queue section.
- fork_job_limit="cpunumber", this option is used to set the number of running grid jobs on the fork machine, allowing a multi-core machine to use some or all of its cores for Grid jobs. The default value is 1.

4.2.3 Implementation details and known limitations

The Fork back-end implements an interface to the “fork” UNIX command which is not a batch system. Therefore the back-end should rather be seen as an interface to the operating system itself. Most of the “batch system values” are determined from the operating system (e.g. cpu load) or manually set in the configuration file.

Since Fork is not a batch system, many of the queue specific attributes or detailed job information is not available. The support for the “Fork batch system” was introduced so that quick deployments and testing of the middleware can be possible without dealing with deployment of a real batch system since fork is available on every UNIX box. The "Fork back-end" is not recommended to be used in production. The back-end by its nature, has lots of limitations, for example it does not support parallel jobs.

4.3 SGE

Sun Grid Engine, formerly known as Codine, is an open source batch system maintained by Sun. It is supported on Linux, and Solaris in addition to a numerous other systems.

4.3.1 Recommended batch system configuration

Set up one or more SGE queues for access by grid users. Queues can be shared by normal and grid users. In case you want to set up more than one ARC queue, make sure that the corresponding SGE queues have no shared nodes among them. Otherwise the counts of free and occupied CPUs might be wrong. Only SGE versions 6 and above are supported.

4.3.2 Relevant arc.conf options

The SGE back-end requires that the following options are specified:

- The SGE batch system back-end is enabled by setting `lrms="sge"` in the `[common]` configuration block.
- `sge_root` must be set to SGE’s install root.
- `sge_bin_path` configuration variable of the `[common]` block must be set to the path of the SGE binaries.
- `sge_cell`, `sge_qmaster_port` and `sge_execd_port` options might be necessary to set in special cases. See the `arc.conf(5)` man page for more details.
- `sge_jobopts` configuration variable of the `[queue]` block can be used to add custom SGE options to job scripts submitted to SGE. Consult SGE documentation for possible options.

Example:

```
lrms="sge"
sge_root="/opt/n1ge6"
sge_bin_path="/opt/n1ge6/bin/lx24-x86"
...
[queue/long]
sge_jobopts="-P atlas -r yes"
```

4.3.3 Implementation details

The SGE back-end’s commands are similar to the PBS commands. These commands are used in the code:
Submit job:

- qsub -S /bin/sh (specifies the interpreting shell for the job)

Get jobs status:
If the job state is not suspended, running or pending then its state is failed.

- qstat -u '*' -s rs (show the running and suspended jobs status)
- qstat -u '*' -s p (show the pending jobs status)
- qstat -j job_id (long job information)
- qacct -j job_id (finished job report)

Job terminating:

- qdel job_id (delete Sun Grid Engine job from the queue)

Queue commands:

- qconf -spl (show a list of all currently defined parallel environments)
- qconf -sql (show a list of all queues)
- qconf -sep (show a list of all licensed processors/slots)
- qstat -g c (display cluster queue summary)
- qconf -sconf global (show global configuration)
- qconf -sq queue_name (show the given queue configuration)

Other:

- qstat -help (show Sun Grid Engine's version and type)

4.3.4 Known limitations

Multi-CPU support is not well tested. All users are shown with the same quotas in the information system, even if they are mapped to different local users. The requirement that one ARC queue maps to one SGE queue is too restrictive, as the SGE's notion of a queue differs widely from ARC's definition. The flexibility available in SGE for defining policies is difficult to accurately translate into NorduGrid's information schema. The closest equivalent of nordugrid-queue-maxqueuable is a per-cluster limit in SGE, and the value of nordugrid-queue-localqueued is not well defined if pending jobs can have multiple destination queues.

5 The ARC GLUE2 implementation

ARC1 provides support for both the classic NorduGrid information schema [**?**] as well as an early minimal GLUE 2 implementation. The output is rendered in XML format.
Currently, the the following GLUE2 entities are included in ARC1:

- ComputingService
- ComputingEndpoint
- ComputingShare

While the following entities are still missing:

- ApplicationEnvironment
- ExecutionEnvironment
- ComputingActivity
- Policy including MappingPolicy / AccessPolicy

The different entities with their attributes are described in their respective sections below.

5.1 ComputingService

Hardcoded Properties:

Property	Value
Capability	`executionmanagement.jobexecution`
Type	`org.nordugrid.arex`
QualityLevel	`development`
Complexity	`endpoint=1,share=N,resource=1`, where N is the number of queues in the configuration file

The following properties referring to job counts are calculated based on information in GM's status files:

Property	Meaning
TotalJobs	number of grid jobs in states other than FINISHED
RunningJobs	number of grid jobs in INLRMS:R state
WaitingJobs	number of grid jobs in INLRMS:Q, ACCEPTED, PREPARING and SUBMIT* states

5.2 ComputingEndpoint

Property	Value
URL	`https://`*`hostname`*`:`*`port`*`/arex`
Technology	`webservice`
Interface	`OGSA-BES`
WSDL	`https://`*`hostname`*`:`*`port`*`/arex/?wsdl`
Semantics	`http://www.nordugrid.org/documents/arex.pdf`
SupportedProfile	`WS-I 1.0` and `HPC-BP`
Implementor	`NorduGrid`
ImplementationName	`ARC1`
ImplementationVersion	`0.9`
QualityLevel	`development`
HealthState	`ok` – no detection of problems yet
ServingState	`draining` if the configuration has allownew=no, `production` otherwise
IssuerCA	obtained from the *hostcert.pem* file
TrustedCA	obtained by scanning the *certificates* directory
Staging	`staginginout`
JobDescription	`ogf:jsdl:1.0`

5.3 ComputingShare

A ComputingShare element is generated for each 'queue' section in the configuration.

Property	Comments
BaseType	`Share`
Name	‘name” configuration option
Description	“comment” configuration option
MappingQueue	“lrms_queue” configuration option
SchedulingPolicy	if MAUI is used, it is set to `fairshare`
ServingState	has the same value as ComputingEndpoint.ServingState

The following properties are taken from the corresponding attributes returned by the LRMS plugin. Units are converted as necessary.

Property	LRMS plugin attribute
MaxCPUTime	maxcputime
MinCPUTime	mincputime
DefaultCPUTime	defaultcput
MaxWallTime	maxwalltime
MinWallTime	minwalltime
DefaultWallTime	defaultwallt

The following properties are directly taken from configuration options.

Property	Configuration option	Comments
MaxTotalJobs	maxjobs	maxjobs is a per cluster limit, grid-manager has no equivalent limit per share
MaxPreLRMSWaitingJobs	maxjobs	
MaxStageInStreams	maxload	maxload is a per cluster limit, grid-manager has no equivalent limit per share
MaxStageOutStreams	maxload	
MaxMemory	nodememory	
MaxDiskSpace	maxdiskperjob	

The following properties are taken from the corresponding LRMS plugin attributes when available:

Property	LRMS plugin attribute
MaxRunningJobs	maxrunning
MaxWaitingJobs	maxqueuable - maxrunning
MaxUserRunningJobs	maxuserrun
MaxSlotsPerJob	maxslotsperjob

The following properties referring to job counts are calculated based on information in GM’s status files:

Property	Meaning
TotalJobs	number of grid jobs in this share in states other than FINISHED
RunningJobs	number of grid jobs in this share in INLRMS:R state
SuspendedJobs	number of grid jobs in this share in INLRMS:S state
WaitingJobs	number of grid jobs in this share in INLRMS:Q and INLRMS:O states
LocalRunningJobs	The total number of jobs running in the associated LRMS queue minus the number of grid-running grid jobs (RunningJobs)
LocalWaitingJobs	The total number of jobs queued in the associated LRMS queue minus the number of grid-queued grid jobs (WaitingJobs)
StagingJobs	number of grid jobs in this share in PREPARING and FINISHING states
PreLRMSWaitingJobs	number of grid jobs in this share in ACCEPTED, PREPARING and SUBMIT* states

The following properties are derived from the corresponding LRMS plugin attributes:

Property	Related LRMS properly
UsedSlots	running
RequestedSlots	queued
FreeSlots	totalcpus - running
FreeSlotsWithDuration	freecpus

5.4 ExecutionEnvironment

Not yet implemented.

5.5 ApplicationEnvironment

Not yet implemented.

5.6 ComputingActivity

Not yet implemented.

5.7 MappingPolicy and AccessPolicy

Not yet implemented.