NORDUGRID

NORDUGRID-TECH-13
30/7/2008

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

Description and developer's guide

Adrian Taga*, Thomas Frågåt†, C.U.Søttrup, Balázs Kónya, Gábor Rőczei

*v.a.taga@fys.uio.no
†thomasf@fys.uio.no

Contents

1 Introduction

This document describes the Advanced Resource Connector[1] batch system back-end infrastrucure. 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 the current solution described in this manual is not a real interface, it is too diffuse to be that.

The batch system back-ends are what tie the ARC grid middleware (through the Grid Manager (GM[2]) and the Information System layer[3]) to the underlying cluster management system or LRMS. The back-ends consist of set of shell and Perl scripts whose role are twofold:

1. to allow the GM to manipulate jobs on the LRMS including job submit, cancel operations and scanning for completed jobs
2. to collect information about jobs, users, 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

As of ARC version 0.6 the the two sets of scripts are located in different directories within the NorduGrid subversion tree[4]. The job control interface scripts can be found under `arc_development_line/grid-manager/LRMS/` while the information collectors are located at `arc_development_line/infosystem/.`

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[4] as well. The archtiecture of ARC1 is different from Classic ARC so 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/`.

Furthermore, the next generation of ARC supports both the classic NorduGrid information schema[3] and currently a minimal set of the GLUE specification version 2.0 schema[7].

The ARC1 back-end architecture and GLUE2 information model implementation is presented in Section TODO: ADD REFERENCE.

2 Job control interface

Job control part of the LRMS interface is handled by the Grid Manager[2]. 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 fit, 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. 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].

2.1 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 a text file containing information about the job. It is called the GRAMi file. It contains such information as the original xRSL option and the job id in the local 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.

2.2 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. Like cancel-LRMS-job it is given the grami file as argument on execution. Submit-LRMS-job then has to set up the session directories, run-time environment and anything else needed. Then it should submit 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 could also be done through an API if the LRMS supports it.

2.3 Scan-LRMS-job

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 should write the exit-code of that job to the file job.{gridid}.lrms_done in the ARC jobstatus directory. Then it should call the gm-kick program the GM will then notice that the job has finished and start finalizing the job.

Generally two approaches are taken to find jobs that are done. One is asking the LRMS. First all started grid jobs are found in the jobstatus directory[‡]. This can be done by checking if the status is INLRMS in the jobs status file[§]. Then the LRMS is asked for the status of those jobs. If they are done they are marked as such and the GM is activated. The problem with this approach is that for most LRMSs the information about finished jobs are only available for a short period after the job finished. Therefore appropriate steps have to be taken if the job is known to have been started but is no longer present in the LRMS. The normal approach is to analyze the jobs status output in the session directory.

The second approach is to parse the LRMSs log files. This method has some drawbacks 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 some where 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 sigle quotes or double quotes. Values without quotes are also accepted. Multi-valued options are not supported currently. Only the last defined value is retained.

`config_update_from_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_update_from_section common
config_update_from_section grid-manager
config_update_from_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 with locally collected information obtained from the batch system. Important to recall that the locally collected information is broader than what the batch-system can offer, information taken from the grid-layer (mostly grid manager), from the front-end machine and from the `arc.conf` configuration file are also needed and used to populate the nordugrid information model[3]. Below only the batch system related information collection is covered. Note that the separation of the different sources is not always that straightforward (e.g. the grid-manager information has some overlap with the batch system info).

The information system interface consists of set of Perl scripts which populate the local lDAP database by generating ldif files. Historically two information provider "frameworks" have been coexisted. As of ARC version 0.6 the old framework is being phased out by the new framework. Briefly about the two frameworks:

- the old framework separated the non-batch system related functionality into the `InfosysCluster.pm`, `InfosysQJU.pm` modules and kept the batch system specific interface within the `cluster-condor.pl`, `queue+jobs+users-condor.pl` (replace condor with your scheduler name) files. There are two templates provided for batch system interface developers: `cluster-gm.pl, queue+jobs+users-gm.pl`
- the new framework originally proposed by Juha Lento[¶] has been brought up-to-date by implementing all the features present in the old framework. The new framework consists of two main scripts, the `cluster.pl` and the `qju.pl`. The non-batch system functionality is implemented by these script (plus some auxiliary scripts). The batch system specific interface is moved to LRMS modules such as `SGE.pm, PBS.pm, Fork.pm`, a template module `Template.pm` is also provided. The main advantage of the new framework is the separation of output generation from the information collection (this will ease the migration to XML output).

The following subsection describes the LRMS interface part of the information system as implemented via the new framework. 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 `LRMS.pm` module, so that it uses the correct Perl module depending on the LRMS type. Short description of the files of the new framework:

- `cluster.pl and qju.pl`: main scripts called to populate the nordugrid information model
- `Shared.pm:` subroutines shared by cluster.pl and cluster.pl
- `LogUtils.pm`: logging utilities
- `LRMS.pm:` defines the LRMS interface (described in the next subsection)
- `Template.pm:` module template for implementing a new LRMS interface
- `SGE.pm, PBS.pm, LSF.pm, Fork.pm, LL.pm, Condor.pm`: actual LRMS modules

3.1 The LRMS Perl module interface

The LRMS module should implement four functions as defined by the `LRMS.pm`: `Cluster_info`, that provides general information about the local cluster, `Queue_info`, that provides information about a specified queue, `Jobs_info` that gives information about a list of specified jobs, and finally `user_info` that gives information about a grid user mapped to a local UNIX account. All the functions are called with a hash containing the contents of arc.conf.

3.1.1 cluster_info

`Cluster_info` is called only with the config hash. It should then return an associative array with the following keys and the appropriate value.

[¶] the original posting of the new framework proposal: http://staff.csc.fi/juha.lento/grid/rewrite.html

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”

3.1.2 queue_info

In addition to the config `queue_info` is called with the name of the queue to get information on. The result should take the following form.

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
mincputime	limit on minimum CPU time for a job in this queue
defaultcput	default CPU time limit for a job in this queue
maxwalltime	limit on maximum wall time for a job in this queue
minwalltime	limit on minimum wall time for a job in this queue
defaultwallt	default wall time limit for a job in this queue
running	number of CPUs used by running jobs (both grid and non-grid) in the queue
queued	CPUs requested by queuing jobs in the queue
totalcpus	number of CPUs available to the queue

3.1.3 jobs_info

Like the rest the first argument is the config hash, then comes the queue name and finally a list of job ids. Once again an associative mapping should be made for each job. The key value pairs are:

key	value
status	LRMS jobstatus mapping‖: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

3.1.4 user_info

`Users_info` is given the config hash, a queue name and a list of local UNIX accounts (UIDs). It should then return an associative array containing for each UID the number of free CPUs available for that local account on the given queue and the number of queued jobs on that queue.

key	value
freecpus	number of freely available CPUs with their time limits in minutes (see nordugrid-authuser-freecpus in[3])
queuelength	estimated queue length for the specified user

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 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 Local Resource Manager System 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 equal than 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's mom config 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 can't 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

The ARC configuration manual contains detailed information on all the ARC configuration parameters[5]and should be consulted as primary source. Below the PBS specific 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 Grid-Manager to determine whether a PBS job is completed. If not specified, the Grid-Manager 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 jobscripts 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 scheduller 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 Condor

The Condor [6] system, developed at the University of Wisconsin-Madison, was initially used to harness free cpu cycles of workstations. Over time it has evolved into a complex system with many grid-oriented features. Condor is available on a large variety of platforms.

4.2.1 Recommended batch system configuration

Install Condor on the Grid Manager (GM) node and configure it as a submit machine. Next, you must add the following to the node's Condor configuration (`CONDOR_IDS` can also be an environment variable):

```
MAIL = <ARC_install_prefix>/libexec/finish-condor-job
CONDOR_IDS = 0.0
```

The `MAIL` attribute will instruct Condor to run the specified program on job completion. The default on Condor is to run /bin/mail to notify the user, but in this case, it is the GM that needs the notification. Therefore, /bin/mail is replaced with a program especially written for talking to the GM.

`CONDOR_IDS` has to be 0.0, so that the above notification program can access the Grid job's session directories (needed to extract the job exit code from the Condor log).

Make sure that no normal users are allowed to submit Condor jobs from this node. For one thing, it would not work for the user, since Condor will try to notify the GM instead of the job owner on job completion. If you don't allow normal user logins on the GM machine, then you don't have to do anything. If you for some reason want to allow users to log into the GM machine, simply don't allow them to execute the condor_submit program. This can be done by putting all local Unix users allocated to the Grid in a single group, e.g. 'griduser', and then setting the file ownership and permissions on condor_submit like this:

```
chgrp griduser $condor_location/bin/condor_submit
chmod 750 $condor_location/bin/condor_submit
```

4.2.2 Relevant arc.conf options

- The Condor batch system back-end is enabled by setting `lrms="condor"` in the `[common]` configuration block.
- `condor_location` configuration variable of the `[common]` block should be set to the Condor install prefix (i.e., the directory containing Condor's bin, sbin, etc).
- `condor_config` configuration variable of the `[common]` block should be set to the value the environment variable `CONDOR_CONFIG` should have (but don't try to use the environment variable directly as `$CONDOR_CONFIG`, since it will probably not be defined when arc.conf is parsed!)
- `condor_rank` configuration variable of the `[common]` block, if defined, will cause the `Rank` attribute to be set in each job description submitted to Condor. Use this option if you are not happy with the way Condor picks out nodes when running jobs and want to define your own ranking algorithm. `condor_rank` should be set to a ClassAd float expression that you could use in the `Rank` attribute in a Condor job description. For example:

  ```
  condor_rank="(1-LoadAvg/2)*(1-LoadAvg/2)*Memory/1000*KFlops/1000000"
  ```

- `condor_requirements` configuration variable of the `[queue]` block defined a subpool of condor nodes. See next section for details.

4.2.3 Implementation details

The job control part of the interface uses the `condor_submit` command to submit jobs. Some of the options used in the job's ClassAd are:

`Requirements` – is used to select the nodes that may run the job. This is how ARC queues are implemented for Condor.

`Periodic_remove` – is used to enforce cputime and walltime limits.

`Log` – the job's condor log file is parsed by the information scripts to find out whether the job was suspended.

The information system component uses the following Condor commands:

`condor_status -long` – for collecting information about nodes

`condor_status -format "%s\n" Machine -constraint '...'` – for listing nodes that make up an ARC queue.

`condor_q -long -global` – for monitoring running jobs.

`condor_history -l jobid` – for collecting information about finished jobs. Further cues are taken from the job's condor log file and the body of the email sent by Condor when a job completes.

4.2.4 Configuring Queues

Condor does not support queues in the classical sense. It is possible, however, to divide the Condor pool in several sub-pools. An ARC 'queue' is then nothing more than a subset of nodes from the Condor pool.

Which nodes go into which queue is defined using the `condor_requirements` configuration option in the corresponding `[queue]` section. It's value must be a well-formed constraint string that is accepted by a `condor_status -constraint '...'` command. Internally, this constraint string is used to determine the list of nodes belonging to a queue. This string can get quite long, so, for readability reasons it is allowed to split it up into pieces by using multiple `condor_requirements` options. The full constrains string will be reconstructed by concatenating all pieces.

Queues should be defined in such a way that their nodes all match the information available in ARC about the queue. A good start is for the `condor_requirements` attribute to contain restrictions on the following: `Opsys`, `Arch`, `Memory` and `Disk`. If you wish to configure more than one queue, it's good to have queues defined in such a way that they do not overlap. In the following example disjoint memory ranges are used to ensure this:

```
[queue/large] condor_requirements="(Opsys == "linux" && (Arch == "intel"
|| Arch == "x86_64")" condor_requirements=" && (Disk > 30000000 && Memory
> 2000)"
[queue/small] condor_requirements="(Opsys == "linux" && (Arch == "intel"
|| Arch == "x86_64")" condor_requirements=" && (Disk > 30000000 && Memory
<= 2000 && Memory > 1000)"
```

Note that '`nodememory`' attribute in arc.conf means the maximum memory available for jobs, while the Memory attribute in Condor is the physical memory of the machine. To avoid swapping (and these are probably not dedicated machines!), make sure that '`nodememory`' is smaller than the minimum physical memory of the machines in that queue. If for example the smallest node in a queue has 1Gb memory, then it would be sensible to use `nodememory="850"` for the maximum job size.

In case you want more precise control over which nodes are available for grid jobs, using pre-defined ClassAds attributes (like in the example above) might not be sufficient. Fortunately, it's possible to mark nodes by using some custom attribute, say `NORDUGRID_RESOURCE`. This is accomplished by adding a parameter to the node's local Condor configuration file, and then adding that parameter to `STARTD_EXPRS`:

```
NORDUGRID_RESOURCE = True
STARTD_EXPRS = NORDUGRID_RESOURCE, $(STARTD_EXPRS)
```

Now queues can be restricted to contain only 'good' nodes. Just add to each `[queue]` section in arc.conf:

```
condor_requirements=" && NORDUGRID_RESOURCE"
```

4.2.5 Known limitations

Only Vanilla universe is supported. MPI universe (for multi-CPU jobs) is not supported. Neither is Java universe (for running Java executables). ARC can only send jobs to Linux machines in the Condor pool, therefore excluding other unixes and Windows destinations. The session directory must be on a network shared directory, visible from all worker nodes.

4.3 LoadLeveler

LoadLeveler(LL), or Tivoli Workload Scheduler LoadLeveler in full, is a parallel job scheduling system developed by IBM.

4.3.1 Recommended batch system configuration

The back-end should work fine with a standard installation of LoadLeveler. For the back-end to report the correct memory usage and cputime spent, while running. LoadLeveler has to be set-up to show this data in the llq command. Normally this is turned off for performance reasons. It is up to the cluster administrator to decide whether or not to publish this information. The back-end will work whether or not this is turned on.

4.3.2 Relevant arc.conf options

Only the two basic LRMS config options are relevant for LoadLeveler:

- The LoadLeveler batch system back-end is enabled by setting `lrms="ll"` in the `[common]` configuration block.
- `ll_bin_path` configuration variable of the `[common]` block must be set to the path of the LoadLeveler binaries.

4.3.3 Implementation details

The LoadLeveler back-end uses LoadLeveler's command line interface(CLI) commands to submit and cancel jobs. All information in the information system is similarly parsed from the output of CLI commands. It does not parse any log files, nor does it use the binary APIs. The reason that the back-end is completely based on the CLI is that the log files are normally kept on another machine than the front end and that the binary API for LL changes quite often. Often with each new version of LL.

4.3.4 Known limitations

There is at the moment no support for parallel jobs on the LoadLeveler back-end.

4.4 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.

4.4.1 Recommended batch system configuration

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

4.4.2 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.

4.4.3 Implementation details and known limitations

The Fork backend implements an interface to the “fork” unix command which is not a batch system. Therefore the backend 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.

Fork is not a batch system, therefore 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 backend is not recommended to be used in production. The backend by its nature, has lots of limitations, for example does not support parallel jobs.

4.5 LSF

4.5.1 Recommended batch system configuration

Set up one or more LSF queues dedicated for access by grid users. All nodes in these queues should have a resource type which corresponds to the one of the the frontend and which is reported to the outside. The resource type needs to be set properly in the ’lsb.queues’ configuration file. Be aware that LSF distinguishes between 32 and 64 bit for Linux. For a homogeneous cluster, the ’type==any’ option is convenient alternative.

Example: In lsb.queues set either of the following:

- RES_REQ = type==X86_64
- RES_REQ = type==any

See the ’-R’ option of the bsub command man page for more explanation.

4.5.2 Relevant arc.conf options

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

- `lrms="lsf"` in the `[common]` configuration block.
- `lsf_bin_path` configuration variable of the `[common]` block must be set to the path of the LSF binaries.
- lsf_profile_path must be set to the filename of the LSF profile that the back-end should use.

Furthermore it is very important to specify the correct architecture for a given queue in arc.conf. Because the architecture flag is rarely set in the xRSL file the LSF back-end will automatically set the architecture to match the chosen queue. LSF’s standard behaviour is to assume the same architecture as the frontend. This will fail for instance if the frontend is a 32 bit machine and all the cluster resources are 64 bit.If this is not done the result will be jobs eing rejected by LSF because LSF believes there are no useful resources available.

4.5.3 Implementation details

The LSF implementation of the back-end are based solely on parsing and running LSF’s command line interface commands. No log files or other methods are used. To get the correct output o any output at all the back-end needs to have an apropriate LSF profile. The path to this profile must be set in arc.conf. It will then be executed by the back-end before running any of LSF’s CLI commands.

4.5.4 Known limitations

Parallel jobs have not been tested on the LSF back-end.

The back-end does not at present support reporting different number of free CPUs per user.

4.6 SGE

Sun Grid Engine, formely known as Codine, is an open source batch system maintained by Sun. Runs on Linux, Solaris.

4.6.1 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 nore 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.

4.6.2 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"
```

4.6.3 Implementation details

The SGE backend'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 cofiguration)

Other:

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

4.6.4 Known limitations

Multi-CPU support is not well tested. All users are shown with the same quotas in infosys, 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 ARC1

This section contains the current implementation of the back-ends in ARC1.

5.1 The Information System

ARC1 provides support for both the classic NorduGrid information schema[3] as well as an early minimal GLUE 2 implementation. The functionality is divided between several information collecting scripts and the output is rendered in XML format. Further details about each script are given in the subsections below. The current GLUE 2 implementation is given in section TODO: CROSS REFERENCE.

5.1.1 The information Collector Scripts

The information collector scripts are divided between several seperate scripts:

- `cluster+qju.pl` - driver for information collection. It calls all other infomation collectors and prints 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 simple 'schema' (not XML Schema)
- `GMJobsInfo.pm` - collects information about jobs from grid manager 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 ...)
- `LRMSInfo.pm` - collects information that is LRMS specific (queues, jobs, local user limits ...)
- `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 NG schema
- `ARC1ClusterInfo.pm` - combines all information about A-REX and produces information structured according to the GLUE2 schema
- `ARC0ClusterSchema.pm` - description of the info produced by ARC0ClusterInfo

- `ARC1ClusterSchema.pm` - description of the info produced by ARC1ClusterInfo

All information about the hosting environment and queues is collected by one invocation of a perl script called `cluster+qju.pl`. The XML output is printed to `STDOUT` and it includes two representations of the output data, namely the classic NorduGrid information schema and an incomplete GLUE2 schema representation.

5.2 The ARC GLUE 2 implementation

Currently, the the following GLUE2 classes are included in ARC1:

- ComputingService
- ComputingEndpoint
- ComputingShare

While the following classes are still missing (TODO: According to our table in wiki, there should be some other sections as well):

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

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

5.2.1 ApplicationEnvironment

LocalID

Attribute value:	LocalID_t
Example:	`TODO`
Related xRSL:	`NA`
UI role:	TODO

Description: An opaque identifier local to the Computing Service.

Name

Attribute value:	String
Example:	`Name: TODO`
Related xRSL:	TODO
UI role:	?

Description: Name of the application environment.

Version

Attribute value:	String
Example:	`Version: TODO`
Related xRSL:	TODO
UI role:	TODO

Description: Version of the application environment.

References

[1] "Advanced Resource Connector middleware for lightweight computational Grids". M.Ellert et al., Future Generation Computer Systems 23 (2007) 219-240.

[2] The NorduGrid Grid Manager And GridFTP Server: Description And Administrator's Manual [NORDUGRID-TECH-2], A.Konstantinov

[3] "The NorduGrid/ARC Information System" [NORDUGRID-TECH-4], B.Kónya

[4] NorduGrid subversion: http://svn.nordugrid.org/trac/nordugrid/browser

[5] ARC configuration document: currently the `arc.conf.template` file.

[6] http://www.cs.wisc.edu/condor/

[7] TODO: ADD REFERENCE