NORDUGRID-TECH-2
30/11/2009

THE NORDUGRID GRID MANAGER AND GRIDFTP SERVER

A. Konstantinov*, D. Cameron

*aleks@fys.uio.no

Contents

1 Introduction

One of the problems the user of widely distributed computing networks faces is different configurations of *Computing Elements* (CE) controlled by different administrators. This makes even initial preparation of a job a non-trivial task. This is especially important in the case of NorduGrid [**?**], where some CEs are not dedicated to NorduGrid and can not be completely reconfigured at low level. Thus some layer capable of performing most of the site-dependent pre- and post-computation of jobs is necessary.

The aim of the *grid-manager* (GM) is to take care of job pre- and post-processing. It provides an interface to stage-in files containing input data and program modules from a wide range of sources and transfer or stage-out output results.

The GM is part of the NorduGrid software, ARC (Advanced Resource Connector). For its connection to other parts please read "An Overview of The NorduGrid Architecture Proposal" [**?**]. The GM uses the Globus Toolkit^{TM} as its underlying software and currently **completely depends** on it.

An additional essential part of the GM is the specialized GridFTP Server (GFS). This server supports a rich subset of the `gsiftp` protocol and has network and local file access parts separated. In the context of the GM its main purpose is to provide control for jobs and controlled access to GM-managed job files through the Grid Security Infrastructure [**?**]. Another option is the Job Control Web Service (JCS) interface implemented as part of the HTTPSD framework [**?**].

All software described here is part of the ARC software toolkit developed by the NorduGrid project **`http://www.nordugrid.org`**.

You should use this document for advanced configuration purposes. It explains the internals of the aforementioned tools and an extended description of configuration options. For installation and configuration refer to other documents available at `http://www.nordugrid.org/papers.html`.

2 Main concepts

A job is a set of input files (which may or may not include executables), a main executable and a set of output files. The process of gathering input files, executing a job, and transferring/storing output files is called a *session*.

Each job is assigned a directory on the CE called the *session directory* (SD). Input files are gathered in the SD. The job can also produce new data files in the SD. The GM does not guarantee the availability of any other places accessible by the job other than the SD (unless such a place is part of the requested Runtime Environment). The SD is also the only place which is controlled by the GM. It is accessible by the user from outside through the GridFTP protocol. Any file created outside the SD is not controlled by the GM. Any exchange of data between client and GM (including also program modules) is done through the GridFTP protocol [**?**] **only**. Each job is associated to an identifier (*jobid*). This is a handle which identifies the job in the GM and the NorduGrid Information System [**?**]. A URL for accessing input/output files is constructed from the base URL available through the NorduGrid Information System as part of the `nordugrid-cluster` under attribute `nordigrid-cluster-contactstring` and *jobid* (jobid corresponds to a SD).

Each job is initiated and controlled through the GFS. All job parameters (not data) are passed to the GM through the GFS in a RSL [**?**] or JSDL-coded [**?**] description (job description – JD). The GM adds its own attributes to Globus RSL [**?**].

3 Input/output data

The main task of the GM is to take care of processing input and output data (files) of the job. Input files are gathered in the SD. There are 2 ways to put files into the SD:

- Downloads initiated by the GM. Such files (name and source) are defined in the JD. It is the sole responsibility of the GM to make sure that a file will be available in the SD.
  The supported sources are at the moment: GridFTP, FTP, HTTP, HTTPS (HTTP over SSLv3), HTTPg (HTTP over GSI) and SRM. Also some indexing service sources are supported: LFC, RC and RLS.
- Upload initiated by the user directly or through the User Interface (UI). Because the SD becomes available immediately at the time of submission of the JD, the UI can (and should) use that to upload data files which

are not otherwise accessible by the GM. An example of such files can be the main executable of the job, files containing the job's options/parameters, etc. These files can (and should) also be specified in the JD.

There is no other reliable way for a job to obtain input data on the CE belonging to NorduGrid. Access to AFS, NFS, FTP, HTTP and any other remote data transport during execution of the job is not guaranteed (at least not by the GM).

The job stores output files in the SD. These files also belong to 2 groups:

- Files which are supposed to be moved to a *Storage Element* (SE) and optionally registered in some *Indexing Service* such as a *Replica Catalog* (RC). The GM takes care of these files. They have to be specified in the JD. If the job fails during any stage of processing no attempt is made to transfer those files to their final destination, unless the option *preserve=yes* is specified in their URLs.
- Files which are supposed to be fetched by the user. The user uses the UI to obtain those files. They **must** also be specified in the JD.

During the transfer of a file, a checksum is calculated on the fly. If the file is being downloaded and the checksum is available from the source, the two checksums are compared and the transfer is failed if they differ. If the file is being uploaded to an indexing service which supports storing checksums, the calculated checksum is sent to the indexing service. For details of the types of checksums supported see [**?**].

If a file transfer fails and the error is judged to be temporary, for example a bad network connection or a remote service is busy, the transfer will be retried after a certain time has passed. The waiting time between each retry increases exponentially with every attempt, and there is a fixed maximum number of attempts before the job is declared as failed.

4 Job flow

From the point of view of the GM a job passes through various states. Figure **??** presents a diagram of the possible states of a job. A user can examine the state of a job by querying the NorduGrid Information System (IS) using the

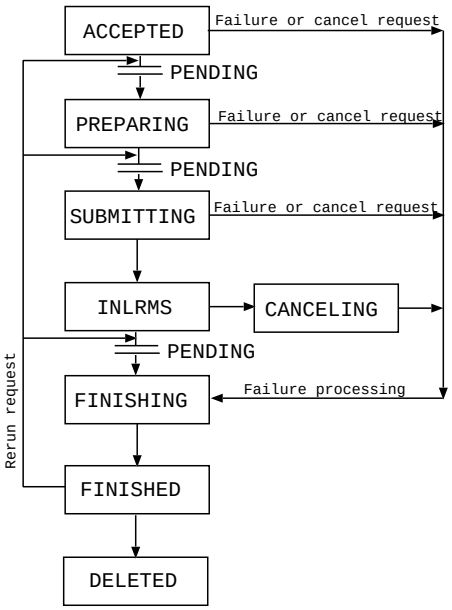

Figure 1: Job states

UI or any other tool. Please remember that the IS can manipulate state names to make them more user friendly and to combine them with states introduced by other parts of whole setup. Another way is to access virtual informational files through the GridFTP interface or to use the query method of the JCS.

The configuration can put limits on the amount of simultaneous jobs in certain states. If such a limit is reached the job stays in its current state waiting for a free slot. Jobs may also wait before moving to the next state due to other

reasons, for example a user-specified earliest job start time or waiting between retries of failed downloads of input files. Any situation where the job cannot move to the next state is presented by prepending the current state name with a **PENDING:** status mark.

Below is a description of all actions taken by the GM at every state:

- **Accepted** - At this state the job has been submitted to a CE but not processed yet. The GM will analyze the JD and move to the next stage. If JD can not be processed the job will be canceled and moved to the state **Finishing**.
- **Preparing** - The input data is being gathered in the SD. The GM is downloading files specified in the JD and waiting for files which should be uploaded by the UI. If all files are successfully gathered the job moves to the next state. If **any** file cannot be downloaded or the UI takes too long to upload a file the job moves to the **Finishing** state (unless the error was temporary, in which case the job goes back to the **Accepted** state and waits for the next retry). It is possible to put a limit on the number of simultaneous **Preparing** jobs. Those jobs out of limit will stay in the previous **Accepted** state with a PENDING mark. Exceptions are jobs which have no files to be downloaded. Those are processed out of limits.
- **Submitting** - This is a point of interaction with the *Local Resource Management System* (LRMS). Several backend batch management systems are supported (e.g. PBS, SGE, Condor). In this state the job is being submitted for execution. If local job submission is successful the job moves to the next state. Otherwise it moves to **Finishing**. It is possible to limit the number of jobs in **Submitting** and following **InLRMS** states.
- **InLRMS** - The job is queued or being executed in the LRMS. The GM takes no actions except waiting until the job finishes.
- **Cancelling** - Necessary action to cancel a job in the LRMS is being taken.
- **Finishing** - The output data is being processed. Specified output data files are moved to the specified SEs and are optionally registered in an indexing service. The user can download data files from the SD using the UI or any other tool. All the files not specified as output files are removed from the SD at very beginning of this state. It is possible to limit the number of simultaneous jobs in this state. If any output file fails to upload but the error is temporary, the jobs moves back to the **InLRMS** state and waits for the next retry.
- **Finished** - No more processing is performed by the GM. The user can continue to download data files from the SD. The SD is kept available for some time (default is 1 week). After that it is moved to the state **Deleted**. The 'deletion' time can be queried from the NorduGrid Information System as attribute `nordugrid-pbs-job-sessiondirerasetime` of `nordugrid-pbs-job`. If the job was moved to **Finished** because of failure, it may be restarted on request of the client. The job is moved to a state previous to the one in which it failed and is assigned the mark PENDING. This is needed in order to not break the configuration limits. The exception to this is a job which failed in **InLRMS** state and is lacking input files specified in JD. Such a job is treated like it failed in the **Preparing** state.
- **Deleted** - The job is moved to this state if the user does not request the job to be cleaned. Only a minimal subset of information about such a job is kept.

In the case of failure, special processing is applied to output files. All specified output files are treated as **downloadable by user** and no files will be uploaded to SEs.

5 URLs

The GM and its components support the following data transfer protocols and corresponding URLs: *ftp, gsiftp, http, httpg, https, lfc, se, srm (v1 and 2.2), rc* and *rls.* For more information please see "Protocols, Uniform Resource Locators (URL) and extensions supported in ARC" [**?**].

6 Internals

6.1 Files

For each local UNIX user listed in the GM configuration a *control directory* exists. In this directory the GM stores information about jobs belonging to that user. Multiple users can share the same *control directory*. To make it easier

to recover in the case of failure, the GM stores most information in files rather than in memory. All files belonging to the same job have names starting with **job.ID.** where ID is the job identifier.

The files in the control directory and their formats are described below:

- *job.ID.status* - current state of the job. It contains one word of text representing the current state of the job. Possible values are :
  - ACCEPTED
  - PREPARING
  - SUBMITTING
  - INLRMS
  - FINISHING
  - FINISHED
  - CANCELING
  - DELETED

See Section **??** for a description of the various states. Additionally each value can be prepended with a prefix "PENDING:" (like PENDING:ACCEPTED, see Section **??**). That is used to show that the job is *ready* to be moved to a next state and it stays in a current state *only* because some limits set in the configuration are exceeded, or it has to wait until a certain time period has elapsed before moving to the next state.

- *job.ID.description* - contains the RSL description of the job.
- *job.ID.local* - information about the job used by the GM. It consists of lines of format *"name = value"* . Not all of them are always available. The following names are defined:
  - *subject* - user subject also known as the distinguished name (DN)
  - *starttime* - the GMT time when the job was accepted represented in Generalized Time format of LDAP
  - *lifetime* - lifetime of the SD after job finishes, in seconds
  - *cleanuptime* - the GMT time when job is to be removed from cluster and SD deleted, in Generalized Time format
  - *notify* - email addresses and flags for sending mail about job-specified status changes
  - *processtime* - the GMT time when to start processing the job in Generalized Time format
  - *exectime* - the GMT time when to start job execution in Generalized Time format
  - *expiretime* - the GMT time when credentials delegated to job expire in Generalized Time format
  - *rerun* - number of retries left to run the job
  - *jobname* - name of the job as supplied by the user
  - *lrms* - name of LRMS on which to run the job
  - *queue* - name of the queue in which to run the job
  - *localid* - job id in LRMS (appears only when the job is in state **InLRMS**)
  - *args* - list of command-line arguments including the executable
  - *downloads* - number of files to download into SD before execution
  - *uploads* - number of files to upload from SD after execution
  - *gmlog* - directory name which holds files containing information about job when accessed through GridFTP interface
  - *clientname* - name and ip address:port of client machine (name is provided by user interface)
  - *clientsoftware* - version of software used to submit job
  - *sessiondir* - SD of job
  - *failedstate* - state at which job failed (available only if it is possible to restart job)
  - *jobreport* - URL of *logger service* used to keep track of executed jobs (one requested by user)

- *transfershare* - name of share used in PREPARING and FINISHING states

This file is filled partially during job submission and fully when the job moves from the **Accepted** to the **Preparing** state.

- *job.ID.input* - list of input files. Each line contains 2 values separated by a space. The first value contains name of the file relative to the SD and the second value is a URL or a file description. Example:

  input.dat gsiftp://grid.domain.org/dir/input_12378.dat

  A URL will be downloaded by the GM and can be an ordinary URL for gsiftp, ftp, http, https, httpg or srm protocols or an indexing service such as rc, rls and lfc. Each URL can contain additional options.

  A file description refers to a file uploaded from the UI and consists of [size][.checksum] where

  size - size of the file in bytes.

  checksum - checksum of the file identical to the one produced by ***cksum*** (1).

  These values are used to verify the transfer of the uploaded file. Both size and checksum can be left out. A special kind of file description *.* is used to specify files which are **not** required to exist.

  This file is used by the '***downloader***' utility. Files with 'url' will be downloaded to the SD and files with 'file description' will simply be checked to exist. Each time a new **valid** file appears in the SD it is removed from the list and *job.ID.input* is updated. Any external tool can thus track the process of collecting input files by checking *job.ID.input.*

- *job.ID.output* - list of output files. Each line contains 1 or 2 values separated by a space. The first value is the name of the file relative to the SD. The second value, if present, is a URL. Supported URLs are the same as those supported by job.ID.input.

  This file is used by the '***uploader***' utility. Files with *url* will be uploaded to SEs and remaining files will be left in the SD. Each time a file is uploaded it is removed from the list and *job.ID.output* is updated. Files not mentioned as output files are removed from the SD at the the beginning of the **Finishing** state.

- *job.ID.failed* - the existence of this file marks the failure of the job. It can also contain one or more lines of text describing the reason of failure. Failure includes a return code different from zero of the job itself.

- *job.ID.errors* - this file contains the output produced by external utilities like ***downloader***, ***uploader***, script for job submission to LRMS, etc on their stderr handle. These are not necessarily errors, but can be just useful information about actions taken during the job processing. In case of problems include content of this file when asking for help. There is special kind of those files with *ID* part made of construct *helper.USERNAME*. Such files contain stderr output of so called *helper* applications described below.

- *job.ID.diag* - information about resources used during execution of the job and other information suitable for diagnostics and statistics. Its format is similar to that of *job.ID.local*. The following names are at least defined:

  - *nodename* - name of computing node which was used to execute the job,
  - *runtimeenvironments* - used runtime environments separated by ';',
  - *exitcode* - numerical exit code of job,
  - *frontend_distribution* - name and version of operating system distribution on frontend computer,
  - *frontend_system* - name of operating system on frontend computer,
  - *frontend_subject* - subject (DN) of certificate representing frontend computer,
  - *frontend_ca* - subject (DN) of issuer of certificate representing frontend computer,

  and other information provided by GNU *time* utility. Note that some implementations of *time* insert unrequested information in their output. Hence some lines can have broken format.

- *job.ID.proxy* - delegated GSI proxy.

- *job.ID.proxy.tmp* - temporary GSI proxy with different unix ownership used by processes run with effective *user id* different from job owner's *id*.

There are other files with names like job.ID.* which are created and used by different parts of the GM. Their presence in the *control directory* can not be guaranteed and can change depending on changes in the GM code.

6.2 Library

There is a library *libarcdata* distributed as part of the GM. *libarcdata* (available only if built using autotools) provides support for moving data between different URLs. Its interface can be found in Appendix **??**.

7 Cache

The GM can cache input files, so that subsequent jobs requiring the same files do not have to download them again. Caching is enabled if one or more cache directories are specified in the configuration file. All input files except files uploaded by the user during job submission are cached by default. This includes excutable files downloaded by the GM. Caching can be explicitly turned off in the job description using the *cache=no* option in input file URLs (for more information on URL options see [**?**]). The disk space occupied by the cache is controlled by removing files in the order of least recent access. For more information on configuration see Section **??**.

7.1 Structure

Cached files are stored in sub-directories under the *data* directory in each main cache directory. Filenames are constructed from an SHA-1 hash of the URL of the file and split into subdirectories based on the two initial characters of the hash.

When multiple caches are used, a new cache file goes to a randomly selected cache, where each cache is weighted according to the size of the file system on which it is located. For example: if there are two caches of 1TB and 9TB then on average 10% of input files will go to the first cache and 90% will go to the second cache.

Some associated metadata including the corresponding URL and an expiry time, if available, are stored in a file with the same name as the cache file, with a *.meta* suffix.

For example, with a cache directory */cache*, the file

lfc://atlaslfc.nordugrid.org//grid/atlas/file1 is mapped to */cache/data/78/f607405ab1df6b647fac7aa97dfb6089c19fb3*

and the file */cache/data/78/f607405ab1df6b647fac7aa97dfb6089c19fb3.meta* contains the original URL and an expiry time if one is available.

At the start of a file download, the cache file is locked, so that it cannot be deleted and so that another download process cannot write the same file simultaneously. This done by creating a file with the same name as the cache filename but with a *.lock* suffix. This file contains the process ID of the process and the hostname of the host holding the lock. If this file is present, another process cannot do anything with the cache file and must wait until the cache file is unlocked (i.e. the *.lock* file no longer exists). The lock has a timeout of one day, so that stale locks left behind by a download process exiting abnormally will eventually be cleaned up. Also, if the process corresponding to the process ID stored inside the lock is no longer running on the host specified in the lock, it is safe to assume that the lock file can be deleted.

7.2 How it works

If a job requests an input file which can be cached or is allowed to be cached, it is stored in the selected cache directory, and depending on the configuration, either the file is copied to the SD or a hard link is created in a per-job directory and a soft link is created in the SD to there. The per-job directories are in the *joblinks* subdirectory of the main cache directory. The former option is advised if the cache is on a file system which will suffer poor performance from a large number of jobs reading files on it, or the file system containing the cache is not accessible from worker nodes. The latter option is the default option. Files marked as executable in the job will be stored in the cache without executable permissions, but they will be copied to the SD and the appropriate permissions applied to the copy.

The per-job directory is only readable by the local user running the job, and the cache directory is readable only by the GM user (usually root). This means that the local user cannot access any other users' cache files. It also means that cache files can be removed without needing to know whether they are in use by a currently running job. **IMPORTANT:** If a cache is mounted from an NFS server and the GM is run by the root user, the server must have

the *no_root_squash* option set for the GM host in the */etc/exports* file, otherwise the GM will not be able to create the required directories.

If the file system containing the cache is full and it is impossible to free any space, the download fails and is retried without using cacheing.

Before giving access to a file already in the cache, the GM contacts the initial file source to check if the user has read permission on the file. In order to prevent repeated checks on source files, this authentication information is cached for a limited time. On passing the check for a cached file, the user's DN is stored in the *.meta* file, with an expiry time equivalent to the lifetime remaining for the user's proxy certificate. This means that the permission check is not performed for this user for this file until this time is up (usually several hours). File modification and validity times from the original source are also checked to make sure the cached file is fresh enough. If the modification time of the source is later than that of the cached file, the file will be downloaded again. The file will also be downloaded again if the modification date of the source is not available, as it is assumed the cache file is out of date. These checks are not performed if the DN is cached and is still valid.

The GM checks the cache periodically. If the used space on the file system containing the cache exceeds the high water-mark given in the configuration file it tries to remove the least-recently accessed files to reduce the cache size to the low water-mark.

7.3 Remote Caches

If a site has multiple GMs running (see Section **??**), a GM can be configured to have its own caches and have read-only access to caches under the control of other GMs (remote caches). An efficient way to reduce network traffic within a site is to configure GMs to be under control of caches on their local disks and have caches on other hosts as remote caches. If a GM wishes to cache a file and it is not available on the local cache, it searches for the file in remote caches. If the file is found in a remote cache, the actions the GM takes depends on the policy for the remote cache. The file may be replicated to the local cache to decrease the load on the remote file system caused by many jobs accessing the file. However, this will decrease the total number of cache files that can be stored. The other policy is to use the file in the remote cache, creating a per-job directory for the hard link in the remote cache. Then the link is created from the session dir to that directory, bypassing the local cache completely. The usual permission and validity checks are performed for the remote file. Note that no creation or deletion of remote cache data is done - cache cleaning is only performed on local caches.

7.4 Cache Administration

The cache is cleaned automatically periodically (every 2 minutes) by the GM to keep the size of each cache within the configured limits. Files are removed from the cache if the total size of the cache is greater than the configured limit. Files which are not locked are removed in order of access time, starting with the earliest, until the size is lower than the configured lower limit. If the lower limit cannot be reached (because too many files are locked, or other files outside the cache are taking up space on the file system), the cleaning will stop before the lower limit is reached.

Since the limits on cache size are given as a percentage of space used on the filesystem on which the cache is located, it is recommended that each cache has its own dedicated file system. If the cache shares space with other data on a file system, changes in the amount of non-cache data will result in changes in the available cache space.

With large caches mounted over NFS and a GM heavily loaded with data transfer processes, cache cleaning can become slow, leading to caches filling up beyond their configured limits. For performance reasons it may be advantageous to disable cache cleaning by the GM, and run the *cache-clean* tool independently on the machine hosting the file system.

Caches can be added to and removed from the configuration as required without affecting any cached data, but after changing the configuration file, the GM should be restarted. If a cache is to be removed and all data erased, it is recommended that the cache be put in a *draining* state until all currently running jobs possibly accessing files in this cache have finished. In this state the cache will not be used by any new jobs, but the hard links in the *joblinks* directory will be cleaned up as each job finishes. Once this directory is empty it is safe to delete the entire cache. See the *cachedir* option in Section **??** for how to set a cache to a draining state.

The following tools (installed in *$NORDUGRID_LOCATION/libexec*) exist to help with administration of the cache:

- *cache-clean* - This tool is used periodically by the GM to keep the size of each cache within the configured limits.
  cache-clean -h gives a list of options. The most useful option for administrators is *-s*, which does not delete anything, but gives summary information on the files in the cache, including information on the ages of the files in the cache.
  It is not recommended to run *cache-clean* manually to clean up the cache, unless it is desired to temporarily clean up the cache with different size limits to those specified in the configuration, or to improve performance by running it on the file system's local node as mentioned above.
- *cache-list* - This tool is used to list all files present in each cache. It simply reads through all the *.meta* files and prints to stdout a list of all URLs stored in each cache and their corresponding cache filename, one per line.

8 Files and directories

8.1 Modules

The GM consists of a few separate executable modules. Those are:

- *grid-manager* - Main module. It is responsible for processing the job, moving it through states, running other modules.
- *downloader* - This is a module responsible for gathering input files in the SD. It processes the *job.ID.input* file and updates it.
- *uploader* - This module is responsible for delivering output files to the specified SEs and registration to indexing services. It processes and updates the *job.ID.output* file.
- *frontend-info-collector* - Utility to gather information about frontend and to put it into the *job.ID.diag* file.
- *gm-kick* - Sends a signal to the GM though a FIFO file to wake it up. It is used to increase the responsiveness of the GM.

The following modules are always run under the Unix account to which the user is mapped.

- *smtp-send.sh* and *smtp-send* - These are the modules responsible for sending e-mail notifications to the user. The format of the mail messages can be easily changed by editing the simple shell script *smtp-send.sh*.
- *submit-*-job* - Here * stands for the name of the LRMS. Curently supported LRMS are PBS/Torque, Condor, SGE, LoadLeveler and SLURM. A *fork* pseudo-LRMS is also supported for testing purposes. This module is responsible for the job submission to the LRMS.
- *cancel-*-job* - This module is for canceling a job which was submitted to LRMS.
- *scan-*-job* -This module is responsible for notifying the GM about completion of the job. Its implementation for PBS system uses server logs to extract information about jobs. If logs are not available it uses the less reliable *qstat* command instead. Other backends use different techniques.

There is also an administrator utility:

- *gm-jobs* - prints information about jobs on the cluster and number of jobs in each state.
  Usage: gm-jobs [-h] [-s] [-l] [-q] [-u uid] [-U name] [-c config_file] [-d control_dir]
  -h - print short help,
  -s - print summary of jobs in each transfer share,
  -l - print more information about each job,
  -q - print only a summary of the GM configuration,
  -u - pretend utility is run by user with id *uid*,
  -U - pretend utility is run by user with name *name*,
  -c - read specified configuration file,
  -d - print information about jobs in specified control directory.

The GM comes with plugins useable for various authorization purposes (see for example the description of the *auth-plugin* command below):

- *inputcheck* - checks if all input files specified in job description are downloadable.
  Usage: inputcheck [-h] [-d debug_level] RSL_fle [proxy_file]
  RSL_file -file with job description,
  proxy_file - credentials proxy.
- *lcas* - executes LCAS plugins on credentials and returns 0 if authorization passed.
  Usage: lcas credentials description [library [db [directory]]]
  credentials - path to file with credentials to authorize,
  description - path to file with job description,
  library - path to LCAS library (full or relative to LCAS directory),
  db - path to LCAS DB file (full or relative to LCAS directory),
  directory - LCAS directory.

8.2 Configuration of the Grid Manager

The GM configuration is done through a single configuration file. The default location of this file is

- */etc/arc.conf*

A different configuration file location can be specified by the environment variable ARC_CONFIG. The configuration file consists of empty lines, lines containing comments (lines starting with #) or configuration commands. It is separated into sections. Each section starts with a string containing

- *[section name/subsection name/subsubsection name]*.

Each section continues until the next section or until the end of the file. The configuration file can have commands for multiple services/modules/programs. Each service has its own section named after it. The GM uses the *[grid-manager]* section. Some services can make use of multiple subsections to reflect their internal modular structure. Commands in section *[common]* apply to all services. Command lines have the format

- *name="arguments string"*.

The following commands are defined:

Global commands (those which affect global parameters of the GM and affect all serviced users, also described in [**?**]):

- ***daemon***=*yes|no* - specifies whether the GM should run in the background after started. Defaults to *yes*.
- ***logfile***=*[path]* - specifies name of file for logging debug/informational output. Defaults to */dev/null* for daemon mode and *stderr* for foreground mode.
- ***user***=*[uid[:gid]]* - specifies user id (and optionally group id) to which the GM must switch after reading configuration. Defaults to *not switch*.
- ***pidfile***=*[path]* - specifies file where process id of GM process will be stored. Defaults to *not write*.
- ***debug***=*number* - specifies level of debug information. More information is printed for higher levels. Currently the highest effective number is 3 and lowest 0. Defaults to 2.

All commands above are generic for every daemon-enabled server in the ARC NorduGrid toolkit (such as GFS and HTTPSD).

- ***joblog***=*[path]* - specifies where to store log file containing information about started and finished jobs.

- ***jobreport****=[URL ... number]* - specifies that GM has to report information about jobs being processed (started, finished) to a centralized service running at the given *URL*. Multiple entries and multiple URLs are allowed. *number* specifies how long (in days) old records are to be kept if they failed to be reported. The last specified value becomes effective.

- ***securetransfer****=yes|no* - specifies whether to use encryption while transferring data. Currently works for GridFTP only. Default is no. It is overridden by values specified in URL options.

- ***passivetransfer****=yes|no* - specifies whether GridFTP transfers are passive. Setting this option to yes can solve transfer problems caused by firewalls. Default is no.

- ***localtransfer****=yes|no* - specifies whether to pass file downloading/uploading task to computing node. If set to yes the GM will not download/upload files, but compose a script which is submitted to the LRMS in order that the LRMS can execute file transfer. This requires the GM and Globus installation to be accessible from computing nodes and environment variables GLOBUS_LOCATION and NORDUGRID_LOCATION to be set accordingly. Default is no.

- ***maxjobs****=[max_processed_jobs [max_running_jobs]]* - specifies maximum number of jobs being processed by the GM at different stages:
  max_processed_jobs - maximum number of concurrent jobs processed by GM. This does not limit the number of jobs which can be submitted to the cluster.
  max_running jobs - maximum number of jobs passed to Local Resource Management System.
  Missing value or -1 means no limit.

- ***maxload****=[max_frontend_jobs [emergency_frontend_jobs [max_transferred_files]]]* - specifies maximum load caused by jobs being processed on frontend:
  max_frontend_jobs - maximum number of jobs in PREPARING and FINISHING states (downloading and uploading files). Jobs in these states can cause a heavy load on the GM host. This limit is applied before moving jobs to PREPARING and FINISHING states.
  emergency_frontend_jobs - if the limit of *max_frontend_jobs* is used only by PREPARING or only by FINISHING jobs, aforementioned number of jobs can be moved to another state. This is used to avoid the case where jobs cannot finish due to a large number of recently submitted jobs.
  max_transfered_files - maximum number of files being transfered in parallel by every job. Used to decrease load on not so powerful frontends.
  Missing value or -1 means no limit.

- ***maxloadshare****=max_share share_type* - specifies a sharing mechanism for data transfer. *max_share* is the maximum number of processes that can run per transfer share and *share_type* is the scheme used to assign jobs to transfer shares. See Section **??** for possible values and more details.

- ***wakeupperiod****=time* - specifies how often external changes are performed (like new arrived job, job finished in LRMS, etc.). *time* is a minimal time period specified in seconds. Default is 3 minutes.

- ***authplugin****=state options plugin* - specifies *plugin* (external executable) to be run every time job is about to switch to *state*. The following states are allowed: ACCEPTED, PREPARING, SUBMIT, FINISHING, FINISHED and DELETED. If the exit code of *plugin* is not 0, the job is canceled by default. *Options* consists of *name=value* pairs separated by commas. The following *name*s are supported:
  timeout - specifies how long in seconds execution of the plugin is allowed to last (mandatory, "*timeout=*" can be skipped for backward compatibility).
  onsuccess, *onfailure* and *ontimeout* - defines action taken in each case (*onsuccess* happens if exit code is 0). Possible actions are:
  pass - continue execution,
  log - write information about result into logfile and continue execution,
  fail - write information about result into logfile and cancel job.

- ***localcred****=timeout plugin* - specifies *plugin* (external executable or function in shared library) to be run every time job has to do something on behalf of local user. Execution of *plugin* may not last longer than *timeout* seconds. If *plugin* looks like *function@path* then function *int function(char*,char*,char*,...)* from shared library *path* is called (*timeout* is not functional in this case). If exit code is not 0, current operation will fail.

- ***norootpower****=yes/no* - if set to yes, all processes involved in job management will use the local identity of a user to which a Grid identity is mapped in order to access the filesystem at the path specified in the ***sessiondir*** command (see below). Sometimes this may involve running a temporary external process.

- ***allowsubmit****=[group ...]* - list of authorization groups of users allowed to submit new jobs while "allownew=no" is active in *jobplugin.so* configuration (see below in section **??**). Multiple commands are allowed.
- ***speedcontrol****=min_speed min_time min_average_speed max_inactivity* - specifies how long or slow data transfer is allowed to be. A transfer is canceled if the transfer rate (bytes per second) is lower than *min_speed* for at least *min_time* seconds, or if average rate is lower than *min_average_speed*, or no data is receved for longer than *max_inactivity* seconds.
- ***copyurl****=template replacement* - specifies that URLs starting from *template* should be accessed in a different way (most probably Unix open). The *template* part of the URL will be replaced with *replacement*. *replacement* can either be a URL or a local path starting from '/'. It is advisable to end template with '/'.
- ***linkurl****=template replacement [node_path]* - mostly identical to *copyurl* but the file will not be copied - instead a soft-link will be created. *replacement* specifies the way to access the file from the frontend, and is used to check permissions. *node_path* specifies how the file can be accessed from computing nodes, and will be used for soft-link creation. If *node_path* is missing - *local_path* will be used instead. Neither *node_path* nor *replacement* should be URLs.

  NOTE: URLs which fit into *copyurl* or *linkurl* are treated as more easily accessible than other URLs. This means if the GM has to choose between several URLs from which should it download input files, these will be tried first.

Per-UNIX user commands:

- ***mail****=e-mail_address* - specifies an email address **from** which notification mails are sent.
- ***defaultttl****=ttl [ttr]* - specifies the time in seconds for the SD to be available after job finishes (*ttl*) and after job is deleted (*ttr*) due to *ttl*. Defaults are 7 days for *ttl* and 30 days for *ttr*.
- ***lrms****=default_lrms_name default_queue_name* - specifies names for the LRMS and queue. A queue name can also be specified in the JD (currently it is not allowed to override the LRMS used using JD).
- ***sessiondir****=path* - specifies the path to the directory in which the SD is created. Multiple session directories may be specified by specifying multiple *sessiondir* commands. In this case jobs are spread evenly over the session directories. If the path is * the default one is used - *$HOME/.jobs* .
- ***cachedir****=path [link_path]* - specifies a directory to store cached data (see **??**). Multiple cache directories may be specified by specifying multiple *cachedir* commands. Cached data will be distributed over multiple caches according to free space in each. Specifying no *cachedir* command or commands with an empty path disables caching. The optional *link_path* specifies the path at which *path* is accessible on computing nodes, if it is different from the path on the GM host. If *link_path* is set to '.' files are not soft-linked, nor are per-job links created, but files are copied to the session directory. If a cache directory needs to be drained, then *cachedir* should specify *"drain"* as the *link_path*.
- ***remotecachedir****=path [link_path]* - specifies caches which are under the control of other GMs, but which this GM can have read-only access to (see Section **??**). Multiple remote cache directories may be specified by specifying multiple *remotecachedir* commands. If a file is not available in paths specified by cachedir, the GM looks in remote caches. *link_path* has the same meaning as in *cachedir*, but the special path "replicate" means files will be replicated from remote caches to local caches when they are requested.
- ***cachesize****=high_mark [low_mark]* - specifies high and low watermarks for space used by each cache, as a percentage of the space on the file system on which the cache directory is located. When *high_mark* is exceeded, files will be deleted to bring the used space down to *low_mark*. It is a good idea to have each cache on its own separate file system. To turn off cache deletion, "cachesize" without parameters can be specified. These cache settings apply to all caches specified by *cachedir* commands.
- ***maxrerun****=number* - specifies maximum number of times job will be allowed to rerun after it has failed in the LRMS. Default value is 2. This only specifies a upper limit. The actual number is provided in the job description and defaults to 0.
- ***maxtransfertries****=number* - specifies the maximum number of times download and upload will be attempted per job (retries are only performed if an error is judged to be temporary). This number must be greater than 0 and defaults to 10.

All per-user commands should be put before the *control* command which initiates the serviced user.

- ***control****=path username [username [...]]* - This option initiates a UNIX user as being serviced by the GM. *path* refers to the control directory (see Section **??** for the description of control directory). If the path is * the default one is used - $HOME/.jobstatus . *username* stands for UNIX name of the local user. Multiple names can be specified. If the name is * it is substituted by all names found in file /etc/grid-security/grid-mapfile (for the format of this file one should study the Globus project [**?**]).
  The special name '.' (dot) can also be used. The corresponding control directory will be used for **any** user. This option should be the last one in the configuration file. The command ***controldir****=path* is also available. It uses the special username '.' and is always executed last independent of its placement in the file.
- ***helper****=username command [argument [argument [...]]]* - associates an external program with a local UNIX user. This program will be kept running under account of the user specified by *username*. Special names can be used: '*' - all names from /etc/grid-security/grid-mapfile, '.' - root user. The user should be already configured with the *control* option (except root, who is always configured). *command* is an executable and *argument*s are passed as arguments to it. The stderr output of an executable (or error message if failed to run an executable) are redirected to a file in the *control directory* called *job.helper.username.errors*.

The following commands are global commands and are specific to the underlying LRMS (PBS in this case).

- ***pbs_bin_path****=path* - path to directory which contains PBS commands.
- ***pbs_log_path****=path* - path to directory with PBS server's log files.
- ***gnu_time****=path* - path to *time* utility.
- ***tmpdir****=path* - path to directory for temporary files.
- ***runtimedir****=path* - path to directory which contains *runtimenvironment* scripts.
- ***shared_filesystem****=yes|no* - if computing nodes have access to the session directory through a shared filesystem like NFS. Corresponds to the environment variable RUNTIME_NODE_SEES_FRONTEND.
- ***nodename****=command* - command to obtain hostname of computing node.
- ***scratchdir****=path* - path on computing node to move session directory to before execution.
- ***shared_scratch****=path* - path on frontend where ***scratchdir*** can be found.

In each command's arguments (paths, executables, ...), the following substitutions can be used:

%R - session root - see command *sessiondir*

%C - control dir - see command *control*

%U - username

%u - userid - numerical

%g - groupid - numerical

%H - home dir - home specified in /etc/passwd

%Q - default queue - see command *lrms*

%L - default lrms - see command *lrms*

%W - installation path - ${NORDUGRID_LOCATION}

%G - globus path - ${GLOBUS_LOCATION}

%c - list of all control directories

%I - job ID (for plugins only, substituted in runtime)

%S - job state (for *authplugin* plugins only, substituted at runtime)

%O - reason (for *localcred* plugins only, substituted at runtime).
Possible reasons are:

new - new job, new credentials

renew - old job, new credentials

write - write/delete file, create/delete directory (through gridftp)

read - read file, directory, etc. (through gridftp)

extern - call external program (grid-manager)

Some configuration parameters can be specified from command line while starting the GM:

grid-manager [-h] [-C level] [-d level] [-c path] [-F] [-U uid[:gid]] [-L path] [-P path]
-h - short help,
-d - debug level,
-L - log file (overwrites value in configuration file),
-P - file containing process id (overwrites value in configuration file),
-U - user and group id to use for running daemon,
-F - do not make process daemon,
-c - name of configuration file,
-C - remove old information before starting: 1 - remove finished jobs, 2 - remove active jobs too, 3 - also remove everything that looks like junk.

8.3 Configuration of the GridFTP Server

The default location of the GFS configuration file is the same as the GM - */etc/arc.conf*. A different configuration file location can be specified by the environment variable ARC_CONFIG. The sections [common] and [gridftpd] are used to configure the GFS. Commands specific to the GFS (in the [gridftpd] section) are described below.

- ***port***=*number* - specifies TCP/IP port number. Default is 2811.
- ***include***=*path* - include contents of another file. Generic commands cannot be specified there.
- ***encryption***=*yes|no* - specifies if server will allow data transfer to be encrypted. Default is yes.
- ***pluginpath***=*path* - specifies the path where plugin libraries are installed.
- ***allowunknown***=*yes|no* - if set to *yes*, clients are not checked against the grid-mapfile. Hence only access rules specified in this configuration file will be applied.
- ***firewall***=*hostname* - use IP address of the *hostname* in response to PASV command instead of IP address of a network interface of the computer. An IP address can be used instead of *hostname*. This command may be useful if the server is situated behind a NAT.
- ***unixgroup***=*group rule* - define local UNIX user and optionally UNIX group to which user belonging to specified authorization *group* is mapped (see Section **??** for definition of group). Local names are obtained from the specified *rule*. If the specified rule could not produce any mapping, the next command is used. Mapping stops at first matched rule. The following rules are supported:
  - ***mapfile*** *file* - the user's subject is matched against a list of subjects stored in the specified file, one per line followed by a local UNIX name.
  - ***simplepool*** *directory* - the user is assigned one of the local UNIX names stored in a file *directory/pool*, one per line. Used names are stored in other files placed in the same *directory*. If a UNIX name was not used for 10 days, it may be reassigned to another user.

- ***lcmaps*** *library directory database* - call LCMAPS functions to do mapping. Here *library* is the path to the shared library of LCMAPS, either absolute or relative to *directory*; *directory* is the path to the LCMAPS installation directory, equivalent to the LCMAPS_DIR variable; *database* is the path to the LCMAPS database, equivalent to the LCMAPS_DB_FILE variable. Each argument except *library* is optional and may be either skipped or replaced with '*'. It is important to ensure that no configured LCMAPS plugin performs switch of local user identity (setuid). That may interfere with way the GFS handles local user identities.
- ***mapplugin*** *timeout plugin* [*arg1* [*arg2* [...]]] - run external *plugin* executable with specified arguments. Execution of *plugin* may not last longer than *timeout* seconds. A rule matches if the exit code is 0 and there is a UNIX name printed on *stdout*. A name may be optionally followed by a UNIX group separated by ':'. In arguments the following substitutions are applied before the plugin is started:
  - %D - subject of users's cerificate,
  - %P - name of credentials' proxy file.

- ***unixvo***=*vo rule* - same as ***unixgroup*** for users belonging to Virtual Organization (VO) *vo*.
- ***unixmap***=*[unixname][:unixgroup] rule* - define a local UNIX user and optionally group used to represent connected client. *rule* is one of those allowed for ***authorization groups*** (see Section **??**) and for ***unixgroup***/***unixvo***. In case of a mapping rule, username is the one provided by the rule. Otherwise the specified *unixname:unixgroup* is taken. Both *unixname* and *unixgroup* may be either omitted or set to '*' to specify missing value.
- ***groupcfg***=*name* - is put into subsections representing a plugin or [group] section and defines if that section is effective. The only unaffected option is ***groupcfg***. If name is empty (or no groupcfg is used at all), following lines apply to all users.

Subsections of the *gridftpd* section specify plugins which serve the virtual FTP path (similar to the UNIX mount command). The name of the subsection is irrelevant but it is useful to use a name related to the plugin, e.g. [gridftpd/jobs] for the *jobplugin*. Inside the subsection, the following commands are supported:

- ***plugin***=*library_name* - use plugin *library_name* to serve virtual path.
- ***path***=*path* - virtual path to serve.

  The GFS comes with 3 plugins: *fileplugin.so, gaclplugin.so* and *jobplugin.so*.

  - *jobplugin.so* supports the following options:
    - ***configfile***=*path* - defines non-standard location of the GM configuration file,
    - ***allownew***=*yes*|*no* - specifies if new jobs can be submitted. Default is *yes*.
    - ***unixgroup***/***unixvo***/***unixmap*** - same options as in the top-level GFS configuration. If the mapping succeeds, the obtained local user will be used to run the submitted job.
    - ***remotegmdirs***=*control_dir session_dir* - specifies control and session directories under the control of another GM to which jobs can be assigned (see Section **??**).
  - *fileplugin.so* supports the following options:
    - ***mount***=*path* - defines the place on local filesystem to which file access operations apply.
    - ***dir***=*path options* - specifies access rules for accessing files in *path* (relative to virtual and real path) and all the files below.
      options is a list of the following keywords:
      - ***nouser*** - do not use local file system rights, only use those specified in this line.
      - ***owner*** - check only file owner access rights.
      - ***group*** - check only group access rights.
      - ***other*** - check only "others" access rights.

      The options above are exclusive. If none of the above are specified, the usual UNIX access rights are applied.
      - ***read*** - allow reading files.
      - ***delete*** - allow deleting files.
      - ***append*** - allow appending files (does not allow creation).

- ***overwrite*** - allow overwriting of existing files (does not allow creation, file attributes are not changed).
- ***dirlist*** - allow obtaining list of the files.
- ***cd*** - allow to make this directory current.
- ***create*** *owner:group permissions_or:permissions_and* - allow creating new files. File will be owned by *owner* and owning group will be *group*. If '*' is used, the user/group to which connected user is mapped will be used. The permissions will be set to *permissions_or* & *permissions_and* (the second number is reserved for future usage).
- ***mkdir*** *owner:group permissions_or:permissions_and* - allow creating new directories.

– *gaclplugin.so* supports the following options:

* ***gacl***=*gacl* - GACL XML.
* ***mount***=*path* - local path served by plugin.

The GACL XML may contain variables which are replaced with values taken from the client's credentials. The following variables are supported:

$subject - subject of user's certificate (DN),
$voms - subject of VOMS[**?**] server (DN),
$vo - name of VO (from VOMS certificate),
$role - role (from VOMS certificate),
$capability - capabilities (from VOMS certificate),
$group - name of group (from VOMS certificate) .

Additionally, the root directory must contain a *.gacl* file with initial ACLs. Otherwise the rule will be "deny all for everyone".

Some configuration parameters can be specified from the command line while starting the GFS:

gridftpd [-h] [-F] [-d level] [-L path] [-P path] [-U uid[:gid]] [-c path] [-p number] [-n number] [-b number] [-B number]

-h - short help,
-F - do not make process daemon,
-d - debug level,
-L - log file (overwrites value in configuration file),
-P - file containing process id (overwrites value in configuration file),
-U - user and group id to use for running daemon,
-c - name of configuration file,
-p - TCP/IP port number,
-n - maximum number of simultaneously served connections,
-b - default size of buffer used for data transfer (default is 64kB),
-B - maximum size of buffer used for data transfer (default is 640kB).

8.4 Using Multiple Grid Managers Under One GFS

For large clusters, using a single machine for all input and output file transfer, as well for the interaction with the LRMS and Information System, can limit the job throughput of the cluster. Running several GMs on separate hosts can help spread the hardware and network load. A single GFS can feed jobs to several GMs, hence a cluster with many GMs still appears as a single site to the outside world. When a job is submitted, the GFS jobplugin assigns a control directory to use for the job from the main *controldir* specified in the GM configuration and any extra *remotegmdirs* specified in the jobplugin configuration. The current algorithm for selecting a control directory uses the modulus of the job id to pick one from an ordered list. In this way the job id defines the control directory, and hence no persistent state is required to be maintained on the server side to associate jobs to control directories.

Each control directory is used by a separate GM, therefore for every *remotegmdirs* command in the GFS configuration, there must be a GM running which defines the corresponding *controldir* and *sessiondir* in its configuration file. Each GM can run independently on its own host, the only requirement is that the control and session directories must be

accessible on the GFS host, and the GFS user must have write access to these directories. It is recommended that these directories are local to the GFS host and exported (via NFS for example) to the other hosts, rather than being on a remote filesystem and exported to the GFS host. This means that any glitches in the network do not cause the GFS host to hang. It is also important that the local user accounts on each host must be synchronised with the GFS host. A GM is not aware that any other GMs are running, as they only see what the GFS decides should go into their own control directory. All communication between the GFS and a GM is through the GM's control directory. Note that in remote control directories there is no way to specify control directories per user, as with the *control* command. Only the *controldir* command can be used and all users will use the same control directory.

One feature of this design is that multiple GMs can share the same LRMS, and hence compete with each other to submit jobs. Therefore any LRMS settings in the configuration files must be carefully matched in order not to bias one GM over another. In most cases each host's configuration file can be identical apart from the control and session directories. Some configuration sections such as the GFS and infosys sections will be ignored by the remote GM hosts as these services are not running.

Cacheing can be set up in a variety of different ways. Each GM can have its own cache, completely independent from any other, which will lead to popular files being replicated in many caches. Or, all caches can be shared with all GMs, which means no replication between caches but heavy intra-site network traffic if the cache file systems are hosted on different hosts. Another option is to give each GM its own cache, but access to the other caches as remote caches (see Section **??**. This avoids replicating files and intra-site network traffic. Replication can still be enabled by specifying "replicate" as the link_path for remote cache dirs, and the advantage of this is that files are copied from the remote cache rather than being downloaded again from source.

When setting up an extra GM, it is important that no other services (GFS, infosys) run on the host. The instances of these services running on the "main" host take care of all the GMs. In other words, the startup scripts for *gridftpd* and *grid-infosys* should be removed from any place where they would be started automatically (usually /etc/init.d/). No host certificates are required for hosts which only run a GM instance. Note that in this multiple GM set up, there is a one to one relationship between control and session directories, hence multiple *sessiondir* commands cannot be used in a GM configuration.

8.5 Transfer Shares

For many jobs, large amounts of input and output data can mean significant time is spent in the PREPARING and FINISHING states gathering input data and writing output data. With FIFO processing, this can lead to one user or group of users blocking the queue for others. The GM implements a sharing system to avoid this problem, by assigning each user or group of users to a "transfer share" and specifying a limit on the number of data transfer processes per share. If one user's jobs' transfer share is using the maximum number of processes and another user submits jobs which are assigned to a different share, the second user's jobs can immediately go to PREPARING, up to the same maximum limit of processes. This means that no matter how many jobs the first user submits, the second user's jobs are not blocked. Assuming the bandwidth from the sources of input data for both users' jobs is similar, the available throughput will then be split evenly between the two users' jobs.

If a limit on the total number of data transfer processes is set in the *maxload* option, the maximum number of processes per transfer share is set by splitting the total maximum evenly among all the shares with jobs in data transfer states, up to the maximum allowed per share.

The scheme used to assign jobs to transfer shares can be set in the *maxloadshare* option. Possible values are:

- *dn* - each job is assigned to a share based on the DN of the user sumbitting the job.
- *voms:vo* - if the user's proxy is a VOMS [**?**] proxy the job is assigned to a share based on the VO specified in the proxy. If the proxy is not a VOMS proxy a default share is used.
- *voms:role* - if the user's proxy is a VOMS proxy the job is assigned to a share based on the role specified in the first attribute found in the proxy. If the proxy is not a VOMS proxy a default share is used.
- *voms:group* - if the user's proxy is a VOMS proxy the job is assigned to a share based on the group specified in the first attribute found in the proxy. If the proxy is not a VOMS proxy a default share is used.

If VOMS is not supported, the *dn* scheme is the only option that should be used, as using a VOMS-based scheme will lead to all jobs being assigned to the default share. The current number of jobs processing and pending processing for each share can be seen with the command *gm-jobs -s*.

Important: If a sharing mechanism based on VOMS is used, server certificates for each supported VO must be installed. It is possible to either download the public key of each VOMS server, or create a special file for each VO containing the server's DN and its CA DN. Instructions are given on NorduGrid's web site at `http://www.nordugrid.org/documents/voms-notes.html`.

8.6 Authorization

Authorization is performed by the GFS by applying a set of rules. Each rule takes one line in the *group* section. For information about supported rules please read "Configuration and authorisation of ARC (NorduGrid) Services" [**?**].

8.7 Directories

The GM is installed into a single installation point referred to as $NORDUGRID_LOCATION and the following sub-directories are used:

$NORDUGRID_LOCATION/bin - tools
$NORDUGRID_LOCATION/libexec - program modules used by the GM
$NORDUGRID_LOCATION/etc - information system configuration files (deprecated)
$NORDUGRID_LOCATION/sbin - daemons
$NORDUGRID_LOCATION/lib - gridftp server plugins and API libraries

The GM also uses the following directories:

- *session root directory* - In this directory the SD is created. There can be multiple directories for the various users specified in the configuration file.
  There are 2 processes which must have permission to create new files and directories in it - the GM and the GFS.
  If any of these processes are run under a dedicated user account, that account needs full permissions in the *session root directory*.
  If these processes are run under the *root* account, the *session root directory* must not be on a filesystem which limits the capabilities of the *root* user, for example an NFS filesystem must use the *no_root_squash* option.
  If there is a need to run processes under the *root* account (to run jobs in the LRMS under different user accounts), but there is no way to provide a suitable *session root directory*, use the *norootpower* command in the configuration of the GM. In this case the GM and GFS will use the identity of the local user to which the Grid identity is mapped to access the *session root directory*. Hence those users will need full access there.
  The GM creates the SD with proper ownership and permissions for the local identity used to run the job. Some filesystems require *executable* permissions on the *session root directory* to be set for the local identity in order that they can access any file or subdirectory there.
  This directory should also be shared among cluster nodes in order for a job to access input files, or some internal mechanism of the LRMS must be used to transfer files to the executing node. For more see Section **??**.
- *control directory* - In this directory the SD stores information about the accepted jobs. Both the GM and GFS processes must have full permissions there.
  A subdirectory called *log* is also created there. It is used to accumulate information about started and finished jobs. This information is periodically sent to the *logger service*.

8.8 LRMS support

The GM comes with support for several LRMS. This number is slowly growing. The features explained below are specific to the **PBS** backend. This support is provided through the *submit-pbs-job*, *cancel-pbs-job*, *scan-pbs-job* scripts. *submit-pbs-job* creates a job script and submits it to PBS. This job script is responsible for moving data between the frontend machine and cluster node (if required) and execution of the actual job. Alternatively it can download input files and upload output if *"localtransfer=no"* is specified in the configuration file.

The behavior of the submission script is mostly controlled using environment variables. Most of them can be specified on the frontend in the GM's environment and overwritten on the cluster nodes through PBS configuration. Some of them may be set in the configuration file too.

PBS_BIN_PATH - path to PBS executables, for example */usr/local/bin*. *pbs_bin_path* configuration command.

PBS_LOG_PATH - path to PBS server logs. *pbs_log_path* configuration command.

TMP_DIR - path to a directory to store temporary files. Default value is */tmp*. *tmpdir* configuration command.

RUNTIME_CONFIG_DIR - path where runtime setup scripts can be found. *runtimedir* configuration command.

GNU_TIME - path to GNU time utility. It is important to give a path to a utility compatible with GNU time. If such a utility is not available, modify *submit-pbs-job* to either reset this variable or change usage of an available utility. *gnu_time* configuration command.

NODENAME - command to obtain name of cluster node. Default is */bin/hostname -f*. *nodename* configuration command.

RUNTIME_LOCAL_SCRATCH_DIR - if defined should contain the path to the directory on computing node which can be used to store a job's files during execution. *scratchdir* configuration command.

RUNTIME_FRONTEND_SEES_NODE - if defined should contain the path corresponding to *RUNTIME_LOCAL_SCRATCH_DIR* as seen on the **frontend** machine. *shared_scratch* configuration command.

RUNTIME_NODE_SEES_FRONTEND - if set to "*no*", this means that the computing node does not share a filesystem with the frontend. In this case the content of the SD is moved to a computing node using means provided by the LRMS. Results are moved back after the job's execution in a similar way. *shared_filesystem* configuration command.

Figures **??**, **??** and **??** present some possible combinations for RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE and explain how data movement is performed. Figures a) correspond to the situation right after all input files are gathered in the session directory and actions taken right after the job script starts. Figures b) show how it looks while the job is running and actions which are taken right after it has finished. Figures c) show the final situation, when job files are ready to be uploaded to external storage elements or be downloaded by the user.

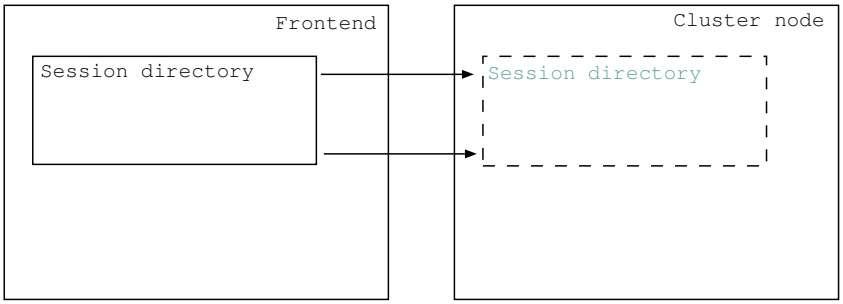

Figure 2: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE undefined. Job is executed in a session directory placed on the frontend.

8.9 Runtime environment

The GM can run specially prepared *BASH* scripts prior to creation of a job script, and before and after execution of the job's main executable. These scripts are requested by the user through the *runtimeenvironment* attribute in RSL and are run with their only argument set equal to '0', '1' or '2' during creation of the job's script, before execution of the main executable and after the main executable finished accordingly. They all are run through BASH's 'source' command, and hence can manipulate shell variables. With argument '0', scripts are run by the GM on the frontend. Some environment variables are defined in that case and can be changed to influence the job's execution later:

- joboption_directory - session directory.
- joboption_arg_# - command and arguments to be executed as specified in RSL.
- joboption_env_# - array of 'NAME=VALUE' environment variables (**not** bash array).

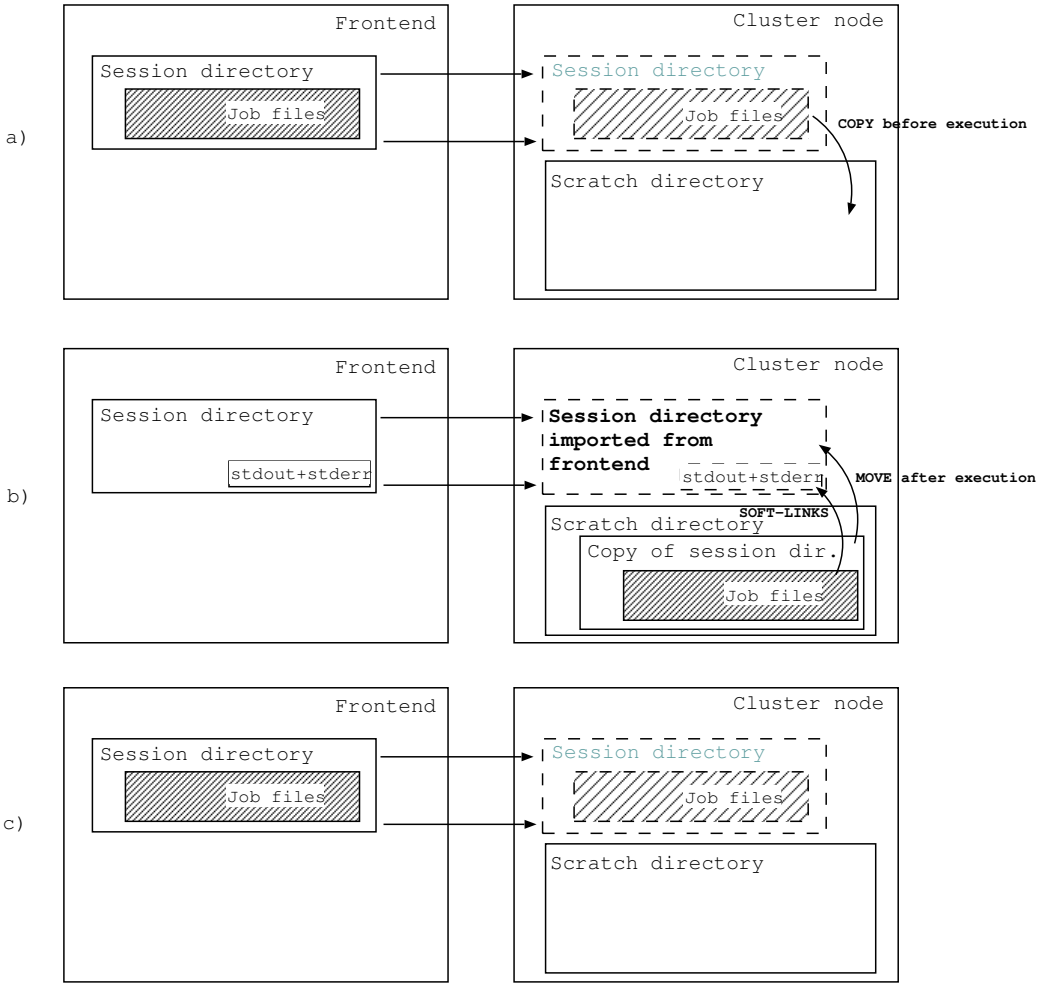

Figure 3: RUNTIME_LOCAL_SCRATCH_DIR is set to a value representing the scratch directory on the computing node, RUNTIME_FRONTEND_SEES_NODE is undefined.

a) After the job script starts all input files are moved to the 'scratch directory' on the computing node.

b) The job runs in a separate directory in 'scratch directory'. Only files representing the job's *stdout* and *stderr* are placed in the original 'session directory' and soft-linked in 'scratch'. After execution all files from 'scratch' are moved back to the original 'session directory'.

c) All output files are in 'session directory' and are ready to be uploaded/downloaded.

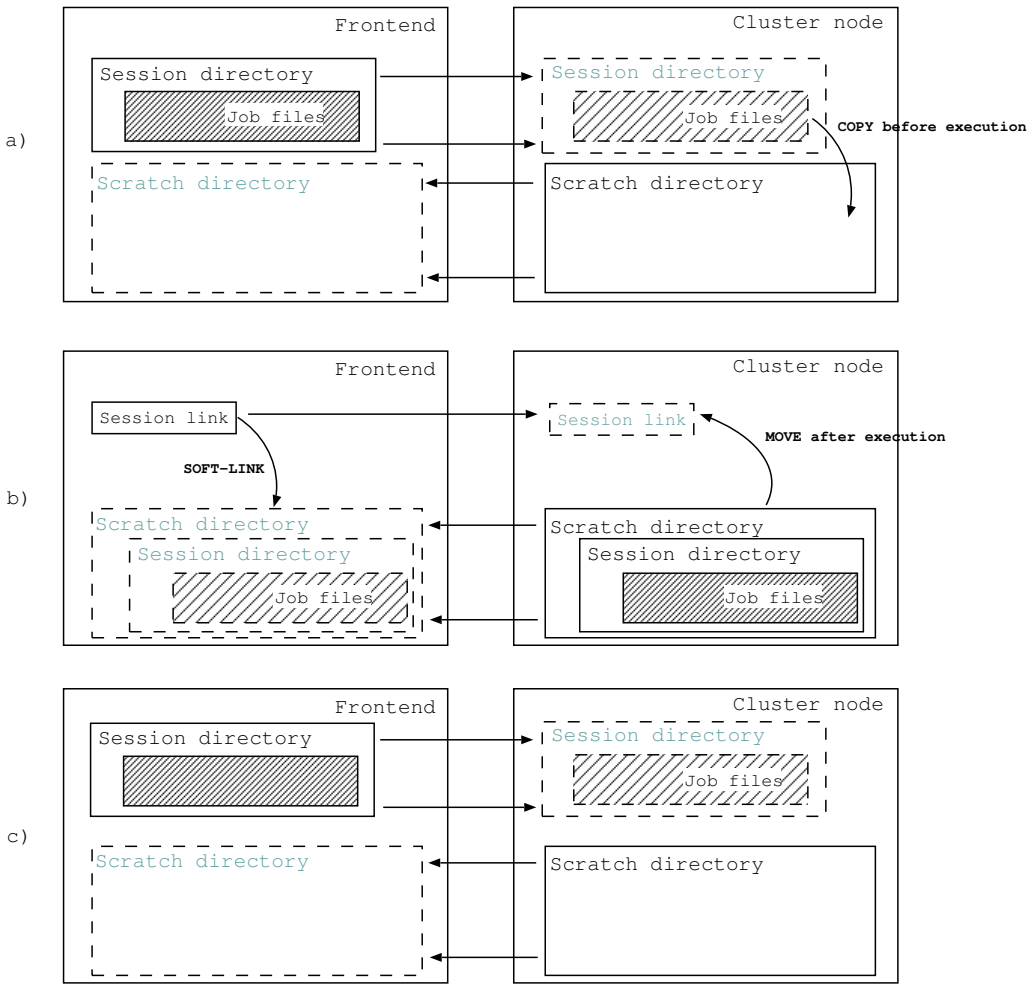

Figure 4: RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are set to values representing the scratch directory on the computing node and a way to access that scratch directory from the frontend respectively.

a) After the job script starts, all input files are moved to 'scratch directory' on the computing node. The original 'session directory' is removed and replaced with a soft-link to a copy of the session directory in 'scratch' as seen on the frontend.

b) The job runs in a separate directory in 'scratch directory'. All files are also available on the frontend through a soft-link. After execution, the soft-link is replaced with the directory and all files from 'scratch' are moved back to the original 'session directory'.

c) All output files are in 'session directory' and are ready to be uploaded/downloaded.

- joboption_runtime_# - array of requested *runtimeenvironment* names (**not** bash array).
- joboption_num - *runtimeenvironment* currently being processed (number starting from 0).
- joboption_stdin - name of file to be attached to stdin handle.
- joboption_stdout - same for stdout.
- joboption_stderr - same for stderr.
- joboption_cputime - amount of CPU time requested (minutes).
- joboption_memory - amount of memory requested (megabytes).
- joboption_count - number of processors requested.
- joboption_lrms - LRMS to be used to run job.
- joboption_queue - name of a queue of LRMS to put job into.
- joboption_nodeproperty_# - array of properties of computing nodes (LRMS specific, **not** bash array).
- joboption_jobname - name of the job as given by user.
- joboption_rsl - whole RSL for very clever submission scripts.
- joboption_rsl_*name* - RSL attributes and values (like joboption_rsl_executable="/bin/echo")

For example *joboption_args* could be changed to wrap the main executable, or *joboption_runtime* could be expanded if the current one depends on others.

With argument '1', scripts are run just before the main executable is run. They are executed on the computing node. Such a script can prepare the environment for some third-party software package. A current directory in that case is one which would be used for execution of the job. The variable HOME also points to that directory.

With argument '2', scripts are executed after the main executable has finished. The main purpose is to clean possible changes done by scripts run with '1' (like removing temporary files). Execution of scripts at that stage also happens on the computing node and is not reliable. If the job is killed by LRMS they most probably will not be executed.

9 Installation

To install the GM as part of an ARC-enabled site please read "NorduGrid ARC server installation instructions" [**?**].

9.1 Requirements

The GM is mostly written using C++. It was tested and should compile on recent enough *Linux* systems using the *gcc* compiler and *GNU make* (gcc versions 2.95, 2.96, 3.2, 3.4 were tested). You will also need *Globus Toolkit*^{TM} version higher than 2.2 installed `http://www-unix.globus.org/toolkit/`.

9.2 Setup of the Grid Manager

For in-depth information about how to properly setup the GM and related software please read "NorduGrid ARC server installation instructions" [**?**]. Follow that manual to install the GM, and configure and run it. Additional tips are described here.

The GM is designed to be able to run both as root and as ordinary user. The name of the user can be specified using the corresponding command in the configuration file. It is better run the GM as root if several users are to be served.

The GM writes debug information into a file `/var/log/grid-manager.log` by default. The file `/var/log/gm-jobs.log` (default path in configuration template, turned off by default) contains information about all started and finished jobs, 2 lines per job (1 when the job is started and 1 after it finished).

9.3 Setup of the GridFTP Server

For in-depth information about how to properly setup the GFS and related software please read "NorduGrid ARC server installation instructions" [**?**]. Follow that manual to install the GFS, configure and run it. Additional tips are described here.

Local file access in the GFS is implemented through plugins (shared libraries). There are 3 plugins provided with the GFS: *fileplugin.so, gaclplugin.so* and *jobplugin.so* . The *fileplugin.so* is intended to be used for plain file access with the configuration sensitive to the user subject and is not necessary for setting up a NorduGrid compatible site. The *gaclplugin.so* uses GACL [**?**] to control access to the local file system. The *jobplugin.so* uses information about jobs being controlled by the GM and provides access to session directories of the jobs owned by the user. It also provides an interface (virtual directory and virtual operations) to submit, cancel, clean, renew credentials and obtain information about the job.

To make GFS to interoperate with other parts of ARC only one *jobplugin.so* needs to be configured. It is advisable to use the template configuration file, and it is possible to leave only the part which configures *jobplugin.so* plugin.

9.4 Usage

Refer to the description of the *User Interface* [**?**] and extensions to RSL [**?**] for using the GM.

9.5 Unix accounts

Both the GM and GFS are designed to be run by the *root* UNIX account and serve multiple local UNIX and global Grid identities. Nevertheless it is possible to use *non-root* accounts to run those services. However this means some functionality loss as described below.

There are no implications from running GFS with *gaclplugin* or *fileplugin* under *non-root* account, as long as only the Grid identity of a user is used and all served files and directories are owned by the server's account.

For a combination of GM and GFS with *jobplugin* both services must be run either by the same account or one of the services must be run under the *root* account. This is needed because services communicate over the local filesystem, hence they must have *full* access to the same set of files.

As long as the GFS with *jobplugin* is run under a non-root account, no mapping from a Grid identity to a local UNIX account takes place. All allowed Grid users are assigned the server's account and are then processed by the GM using the same account. The only way to overcome this limitation is to run one GFS per local account with proper access control configured.

Because the GM has to represent the user's local account while communicating with the LRMS, it can serve only the account it is run under (unless it is run under the *root* account, of course). As in case of the GFS, multiple instances of GM may be run, one per local account. This solution causes another implications however. The GM loses the possibility to share cached files among serviced users. It is also not possible to control the load on a frontend by limiting the number of simultaneously running *downloader* and *uploader* modules.

One has also to take into account that the private part of the GSI infrastructure (the private key of a host at least) has to be duplicated for every account used to run the GFS.

A Job control over jobplugin.so

A.1 Virtual tree

Under the mount point of the jobplugin, the gridftp client can see directories representing jobs belonging to the user who started the client. There is one directory per job. The directory names correspond to job identifiers. These directories are directly connected to the session directories of jobs and contain the same files and subdirectories, unless the job's session directory is moved to the computing node. In that case the directories only contain files with redirected stdout and stderr as specified in the xRSL.

If the job's xRSL has *gmlog* specified, the job's directory also contains a virtual subdirectory with the same name, which contains files with information about the job as created by the GM. The most important are 'errors' and 'status'.

'errors' contains the stderr output of separate modules run by the GM in order to process the job (downloader, uploader, job submission to LRMS). 'status' contains one word representing the state of the job.

Also under the mount point there is an additional directory named "new", used to submit new jobs. Another directory "info" contains subdirectories named after job ids. Those subdirectories contain files with information about the job identical to those in the subdirectory specified through *gmlog*.

A.2 Submission

Each xRSL put into directory "new" is accepted as the job description. The jobplugin parses it and the client receives a positive response if there are no errors in the request.

The job is assigned an identifier and a corresponding directory is created. If the job's description contains input files which should be delivered from the client's machine, the client must upload them to that directory with the specified names.

As each job has an identifier, there should be a way for the client to obtain it. Prior to providing the xRSL, the client sends the command CWD to change the current directory to "new". In this way the job's identifier is reserved, a new directory corresponding to that identifier is created and the client is redirected to it (as specified in the FTP protocol). The job description put into "new" will use the reserved identifier.

A.3 Actions

Various actions to affect the processing of an existing job are performed by uploading xRSL files into directory "new". The content of the xRSL may consist of only 2 parameters - action for the *action* to be performed, and *jobid* to identify the job to be affected. The rest of the parameters are ignored.

Currently supported actions are:

cancel to cancel job

clean to remove job from the computing resource

renew to renew credentials delegated to job

restart to restart job after failure at some phase

It is also possible to perform some actions by using shortcut FTP operations as described below.

A.3.1 Cancel

Job is canceled by performing the DELE (delete file) command on the directory representing the job. It can take some time (a few minutes) before the job is actually canceled. Nevertheless the client gets a response immediately.

A.3.2 Clean

The job's content is cleaned by performing the RMD (remove directory) command on the directory representing the job. If the job is in the "FINISHED" state it will be cleaned immediately. Otherwise it will be cleaned after it reaches the state "FINISHED".

A.3.3 Renew

If the client requests CWD to the session directory, credentials passed during authentication are compared to the current credentials of the job. If the validity time of the new credentials is longer, the job's credentials are replaced with new ones.

B Library *libarcdata*

libarcdata is now part of the *libngui* library. Its functions are declared in a header file *ngdata.h*. They correspond to the ng* utilities for data handling - *arcacl*, *arccp*, *arcls*, *arcrm*, *arctransfer*. It consists of the following functions:

```
void arcacl(const std::string& file_url, const std::string& command, int timeout = 0);

void arcregister (const std::string& source_url, const std::string& destination_url,
                  bool secure = false, bool passive = true, bool force_meta = false,
                  int timeout = 0);

void arccp (const std::string& source_url, const std::string& destination_url,
            const std::string& cache_dir, bool secure = false, bool passive = true,
            bool force_meta = false, int recursion = 0, bool verbose = false, int timeout = 0);

void arcls(const std::string& dir_url, bool show_details = false, bool show_urls = false,
           bool show_meta = false, int recursion = 0, int timeout = 0);

void arcrm(const std::string& file_url, bool errcont = false, int timeout = 0);

void arctransfer(const std::string& destination, std::list<std::string>& sources,
                 int timeout = 0);
```

This library also contains C++ classes used by *ng** data management utilities. Those are described in "ARC::DataMove Reference Manual" [**?**].

C Error messages of GM

If a job has not finished successfully, the GM writes one or more lines of text into the file *job.ID.failed* describing reasons for the failure. Possible reasons include those caused by the GM itself:

Error string	*Reason/description*
Internal error	Error in internal algorithm
Internal error: can't read local file	Error manipulating files in the control directory
Failed reading local job information	-//-
Failed reading status of the job	-//-
Failed writing job status	-//-
Failed during processing failure	-//-
Serious troubles (problems during processing problems)	-//-
Failed initiating job submission to LRMS	Could not run backend executable to pass job to LRMS
Job submission to LRMS failed	Backend executable supposed to pass job to LRMs returned non-zero exit code
Failed extracting LRMS ID due to some internal error	Output of Backend executable supposed to contain local ID of passed job could not be parsed
Failed in files upload (post-processing)	Failed to upload some or all output files
Failed in files upload due to expired credentials - try to renew	Failed to upload some or all output files most probably due to expired credentials (proxy certificate)
Failed to run uploader (post-processing)	Could not run *uploader* executable
uploader failed (postprocessing)	Generic error related to *uploader* component
Failed in files download (pre-processing)	Failed to upload some or all input files
Failed in files download due to expired credentials - try to renew	Failed to download some or all input files most probably due to expired credentials (proxy certificate)
Failed to run downloader (pre-processing)	Could not run *downloader* executable

downloader failed (preprocessing)	Generic error related to *downloader* component
User requested to cancel the job	GM detected external request to cancel this job, most probably issued by user
Could not process RSL	Job description could not be processed to syntax errors or missing elements
User requested dryrun. Job skiped.	Job description contains request not to process this job
LRMS error: (CODE) DESCRIPTION	LRMS returned error. CODE is replaced with numeric code of LRMS, and DESCRIPTION with textual description
Plugin at state STATE failed: OUTPUT	External plugin specified in GM's configuration returned non-zero exit code. STATE is replcaced by name of state to which job was going to be passed, OUTPUT by textual output generated by plugin.
Failed running plugin at state STATE	External plugin specified in GM's configuration could not be executed.

Provided by downloader component (URL is replcaced by source of input file, FILE by name of file):

Error string	*Reason/description*
Internal error in downloader	Generic error
Input file: URL - unknown error	Generic error
Input file: URL - unexpected error	Generic error
Input file: URL - bad source URL	Source URL is either malformed or not supported
Input file: URL - bad destination URL	Shouldn't happen
Input file: URL - failed to resolve source locations	File either not registred or other problems related to Data Indexing service.
Input file: URL - failed to resolve destination locations	Shouldn't happen
Input file: URL - failed to register new destination file	Shouldn't happen
Input file: URL - can't start reading from source	Problems related to accessing instance of file at Data Storing service.
Input file: URL - can't read from source	-//-
Input file: URL - can't start writing to destination	Access problems in a session directory
Input file: URL - can't write to destination	-//-
Input file: URL - data transfer was too slow	Timeouted while trying to download file
Input file: URL - failed while closing connection to source	Shouldn't happen
Input file: URL - failed while closing connection to destination	Shouldn't happen
Input file: URL - failed to register new location	Shouldn't happen
Input file: URL - can't use local cache	Problems with GM cache
Input file: URL - system error	Operating System returned error code where unexpected
Input file: URL - delegated credentials expired	Access to source requires credententials and they are either outdated or missing (not delegated).
User file: FILENAME - Bad information about file: checksum can't be parsed.	In job description there is a checksum provided for file uploadable by user interface and this record can't be interpreted.

User file: FILENAME - Bad information about file: size can't be parsed.	In job description there is a size provided for file uploadable by user interface and this record can't be interpreted.
User file: FILENAME - Expected file. Directory found.	Instead of file uploadable by user interface GM found directory with same name in a session directory.
User file: FILENAME - Expected ordinary file. Special object found.	Instead of file uploadable by user interface GM found special object with same name in a session directory.
User file: FILENAME - Delivered file is bigger than specified.	The size of file uploadable by user interface is bigger than specified in job description.
User file: FILENAME - Delivered file is unreadable.	GM can't check user uploadable file due to some internal error. Most probably due to improperly configured local permissions.
User file: FILENAME - Could not read file to compute checksum.	GM can't read user uploadable file due to some internal error. Most probably due to improperly configured local permissions.
User file: FILENAME - Timeout waiting	GM waited for user uploadable file too long.

Provided by uploader component (URL is replcaced by destination of output file) :

Error string	*Reason/description*
Internal error in uploader	Generic error
Output file: URL - unknown error	Generic error
Output file: URL - unexpected error	Generic error
User requested to store output locally URL	Destination is URL of type *file*.
Output file: URL - bad source URL	Shouldn't happen
Output file: URL - bad destination URL	Destination URL is either malformed or not supported
Output file: URL - failed to resolve source locations	Shouldn't happen
Output file: URL - failed to resolve destination locations	Problems related to Data Indexing service.
Output file: URL - failed to register new destination file	-//-
Output file: URL - can't start reading from source	User request to store output file, but there is no such file or there are problems accessing session directory
Output file: URL - can't start writing to destination	Problems with Data Storing services
Output file: URL - can't read from source	Problems accessing session directory
Output file: URL - can't write to destination	Problems with Data Storing services
Output file: URL - data transfer was too slow	Timeout during transfer
Output file: URL - failed while closing connection to source	Shouldn't happen
Output file: URL - failed while closing connection to destination	Shouldn't happen
Output file: URL - failed to register new location	Problems related to Data Indexing service.
Output file: URL - can't use local cache	Shouldn't happen
Output file: URL - system error	Operating System returned error code where unexpected
Output file: URL - delegated credentials expired	Access to destination requires credententials and they are either outdated or missing (not delegated).

Coming from LRMS (PBS) backend:

Error string	*Reason/description*
Submission: Configuration error.	
Submission: System error.	
Submission: Job description error.	
Submission: Local submission client behaved unexpectedly.	
Submission: Local submission client failed.	