
NORDUGRID-TECH-2
5/12/2008

# THE NORDUGRID GRID MANAGER AND GRIDFTP SERVER

A. Konstantinov*, D. Cameron

*aleks@fys.uio.no

# Contents


**1 Introduction** **3**

**2 Main concepts** **3**

**3 Input/output data** **3**

**4 Job flow** **4**

**5 URLs** **5**

**6 Internals** **5**

6.1 Files . . . 5

6.2 Library . . . 8

**7 Cache** **8**

7.1 Structure . . . 8

7.2 How it works . . . 8

7.3 Administration tools . . . 9

**8 Files and directories** **9**

8.1 Modules . . . 9

8.2 Configuration of the Grid Manager . . . 10

8.3 Configuration of the GridFTP Server . . . 14

8.4 Authorization . . . 17

8.5 Directories . . . 17

8.6 LRMS support . . . 17

8.7 Runtime environment . . . 18

**9 Installation** **21**

9.1 Requirements . . . 21

9.2 Setup of the Grid Manager . . . 21

9.3 Setup of the GridFTP Server . . . 22

9.4 Usage . . . 22

9.5 Unix accounts . . . 22

**A Job control over jobplugin.so** **22**

A.1 Virtual tree . . . 22

A.2 Submission . . . 23

A.3 Actions . . . 23

A.3.1 Cancel . . . 23

A.3.2 Clean . . . 23

A.3.3 Renew . . . 23

**B Library *libarcdata*** **24**

**C Error messages of GM** **24**

# 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 [1], 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 wide range of sources and transfer or store output results.

The GM is part of the NorduGrid software (codename ARC - Advanced Resource Connector). For its connection to other parts please read "An Overview of The NorduGrid Architecture Proposal" [2]. It uses the Globus Toolkit$^{TM}$ as it's 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 access to the job files based on the user subject and job owner. Another option is the Job Control Web Service (JCS) interface implemented as part of the HTTPSD framework [3].

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 is supposed to produce new data files also 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 [4] **only**. 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 associated to an identifier (*jobid*). This is a handle which identifies the job in the GM and the NorduGrid Information System [5].

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 [6] or JSDL-coded [7] description (job description – JD). The GM adds its own attributes to Globus RSL [8].

# 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 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 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 job fails during any stage of processing no attempt is taken to transfer those files to their final destination, unless there is the option *preserve=yes* specified in their URLs.
- Files which are supposed to be fetched by the user. The user runs the UI to obtain those files. They **must** also be specified in the JD.

# 4 Job flow

From the point of view of the GM a job passes through various states. Figure 1 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

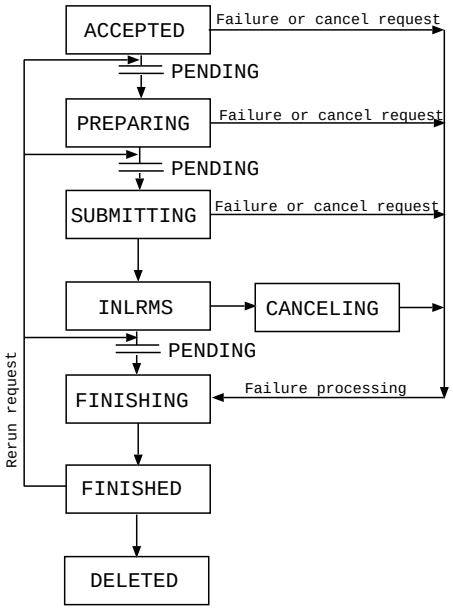


Figure 1: Job states

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. This situation 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 are supposed to be downloaded by the UI. If all files are successfully gathered the job moves to the next state. If **any** file can't be downloaded or it takes UI too long to upload a file - the job moves to **Finishing** state. 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). At the moment PBS is supported best and corresponding backends are provided with the default installation. In this state the job is being submitted for execution. If the 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 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 number of simultaneous jobs in this state.
- **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 one which failed and is assigned the mark PENDING. This is needed in order to not break the configuration limits. An exception is a job failed in **InLRMS** state and lacking input files specified in JD. Such 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**. No files will be moved to the SE.

# 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" [9].

# 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 same job have names starting with **job.ID.** here 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 4 for a description of the various states. Additionally each value can be prepended with a prefix "PENDING:" (like PENDING:ACCEPTED, see Section 4). 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.

- *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* - time period to live for the SD after job finished in seconds
  - *cleanuptime* - the GMT time when job to be removed from cluster and SD deleted in Generalized Time format
  - *notify* - email addresses and flags to send mail to 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 to run the job at
  - *queue* - name of the queue to run the job at
  - *localid* - job id in LRMS (appears only then the job is at 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)

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. First value contains name of the file relative to the SD. Second value is a URL or a file description. Example:

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

  *url* - ordinary URL for gsiftp, ftp, http, https, httpg or srm protocols with the addition of '**replica catalog url**' (RC URL), '**replica location service url**' (RLS URL) and '**LCG file catalog url**' (LFC URL).
  Each URL can contain additional options.

  *file description* - [size][.checksum].

  *size* - size of the file in bytes.

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

  Both size and checksum can be left out. 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. 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 SE 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 the 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.
- *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 the 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 B.

# 7 Cache

The GM can cache input files. Caching is enabled if one or more cache directories are specified in the configuration file. The GM does not cache files marked as executable in a job. Caching can also be explicitly turned off by the user for each file by using the *cache=no* option in the URL (for URL options read "Protocols, Uniform Resource Locators (URL) and extensions supported in ARC" [9]). The disk space occupied by the cache is controlled by removing old files. For more information see Section 8.2.

## 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. This enables the cached files to be evenly split over a number of subdirectories. If more than one cache directory is used the initial letter of the hash also determines which cache is to be used. The algorithm is simply to use the cache directory at index *i* in the list of directories, where *i* is found from the initial letter of the hash mod the number of caches. This limits the maximum number of cache directories to 16, as SHA-1 hashes use the character set [*0-9,a-f*]. In the extremely unlikely event of a collision between two URLs having the same SHA-1 hash, caching will not be used for the second file.

There is no indexing system for the cache, and any cache filename can be easily determined from a URL by using the same hashing algorithm as the GM, e.g. the standard command line tool *sha1sum.* The cache directory used by a file can be determined from the initial letter of the hash and the order of cache directories in the configuration file. Some associated metadata (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. 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. Also file creation or validity times from the original source are checked to make sure the cached file is fresh enough. If it is impossible to obtain creation and invalidation times for the file, it is invalidated 24 hours after download.

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 size to the low water-mark.

### 7.3 Administration tools

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

- *cache-clean* - This tool is used periodically (every 2 minutes) by the GM to keep the size of each cache within the configurated limits. It removes files from the cache if the total size of the cache is greater than the configured limit. It will attempt to remove files which are not locked 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 tool will exit before the lower limit is reached.
  *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.
- *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 signal to the GM though 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. Also *fork* pseudo-LRMS is supported for testing purposes. This module is responsible for the job submission to the LRMS.
- *cancel-*-job* - This one is for canceling a job which was submitted to LRMS.
- *scan-*-job* -This shell script is responsible for notifying the GM about completion of the job. It's 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 list of jobs available on the cluster and amount of jobs in every state.
  Usage: gm-jobs [-a] [-h] [-l] [-u uid] [-U name]
  -a - print information about all jobs on this site (including jobs handled by other GMs),
  -h - print short help,
  -l - print more information about each job,
  -u - pretend utility is run by user with id *uid*,
  -U - pretend utility is run by user with name *name*.

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. Historically the GM supports 2 kinds of configuration files. For the old one it looks in the following places:

- *$NORDUGRID_LOCATION/etc/grid-manager.conf*
- */etc/grid-manager.conf*

And for the new one in

- */etc/arc.conf*

The old configuration file consists of empty lines, lines containing comments (line starts with #) or configuration commands. Blank spaces in arguments must be escaped using '\' or arguments must be enclosed in '"'. Command lines start with a command, followed by arguments separated from the command and each other by spaces.

The new configuration file can also contain empty lines and comments starting from #. 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. One 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 names are the same as in the old configuration file. The *argument string* consists of the same arguments as in the old format, and they must obey the same rules.

Both these files support almost the same commands. The following commands are defined (examples are given for new format):

Global commands (those which affect global parameters of the GM and affect all serviced users, also described in [10]):

- ***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 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.

- ***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 ***session*** command (see below). Sometimes this may involve running 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 8.3). 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. The *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 was deleted (*ttr*) due to *ttl*. Defaults are 7days for *ttl* and 30 days for *ttr*.

- ***defaultlrms***=*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 used LRMS using JD). In the new configuration file this command is called ***lrms***.

- ***session***=*path* - specifies the path to the directory in which the SD is created. If the path is * the default one is used - *$HOME/.jobs* . In the new configuration file this command is called ***sessiondir***.

- ***cachedir***=*path [link_path]* - specifies a directory to store cached data (see 7). Multiple cache directories may be specified by specifying multiple *cachedir* commands. Cached data will be distributed evenly over the caches. 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. In the old configuration file this command is called ***cache***.

- ***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.

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 6 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 [11]).
  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. In the new 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 following commands are global commands supported only in the new configuration file. Most of them are specific to the underlying LRMS (PBS in this case) and are passed in environment variables if the old configuration file is used.

- ***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 *session*

%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 'defaultlrms'

%L - default lrms - see command 'defaultlrms'

%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 */etc/arc.conf* or *$NORDUGRID_LOCATION/etc/gridftpd.conf*. The format of this configuration file is similar to that of the GM. It also supports the generic commands described at the beginning of the previous Section 8.2. In the new format, sections [common] and [gridftpd] are used. Commands specific to the GFS 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 8.4 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 '*'.
  - ***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 8.4) 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. In the old format it selects the group to which all following lines apply. 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. In the old format this section starts with the command ***plugin*** *path library_name* and ends with keyword ***end***. 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* does not require any specific options in the old configuration format. It reads the configuration file of the GM located at the standard place as specified in the Section 8.2. The following options are supported:
    - ***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.

- *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* does not have any options in the old configuration format. The first line of this plugin's configuration contains the local path (root directory) served by it. The rest until the keyword ***end*** contains GACL [12] XML used to setup initial access rules for every newly created file and directory. If the GACL XML is empty then there will be no default ACLs created for new files and directories. This means that the ACL of the parent directory will be used.
  For the new configuration format, the following options are supported:
    - ***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[13] 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] [-p number] [-n number] [-b number] [-B number] [-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,
*-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 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" [10].

## 8.5 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 - configuration files, deprecated, central configuration file is used by deault
$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 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 8.6.
- *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*.

## 8.6 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 2, 3 and 4 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.

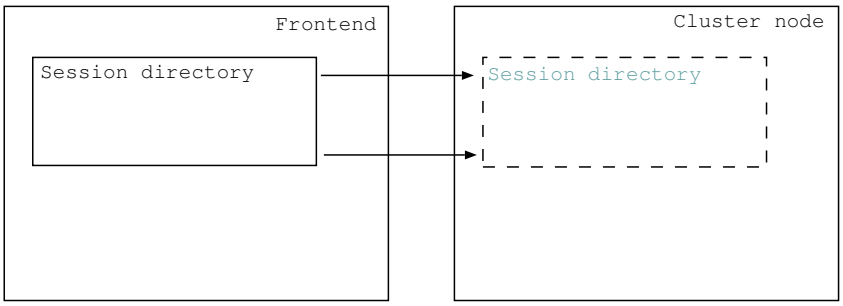


Figure 2: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE undefined. Job is executed in a session directory placed on the frontend.

## 8.7 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.

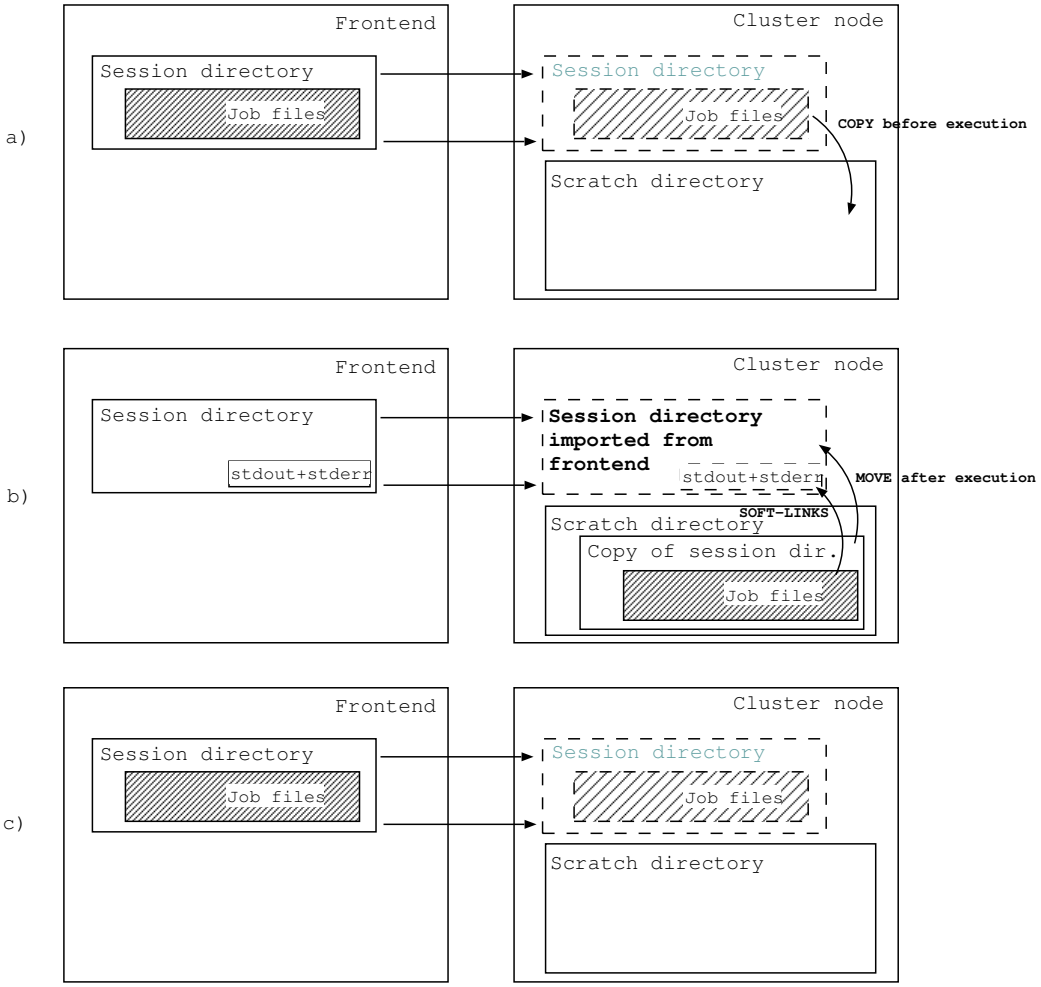


Figure 3: RUNTIME_LOCAL_SCRATCH_DIR is set to a value representing the scratch directory on the computing node, RUNTIME_FRONTEND_SEES_NODE is undefined.

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.

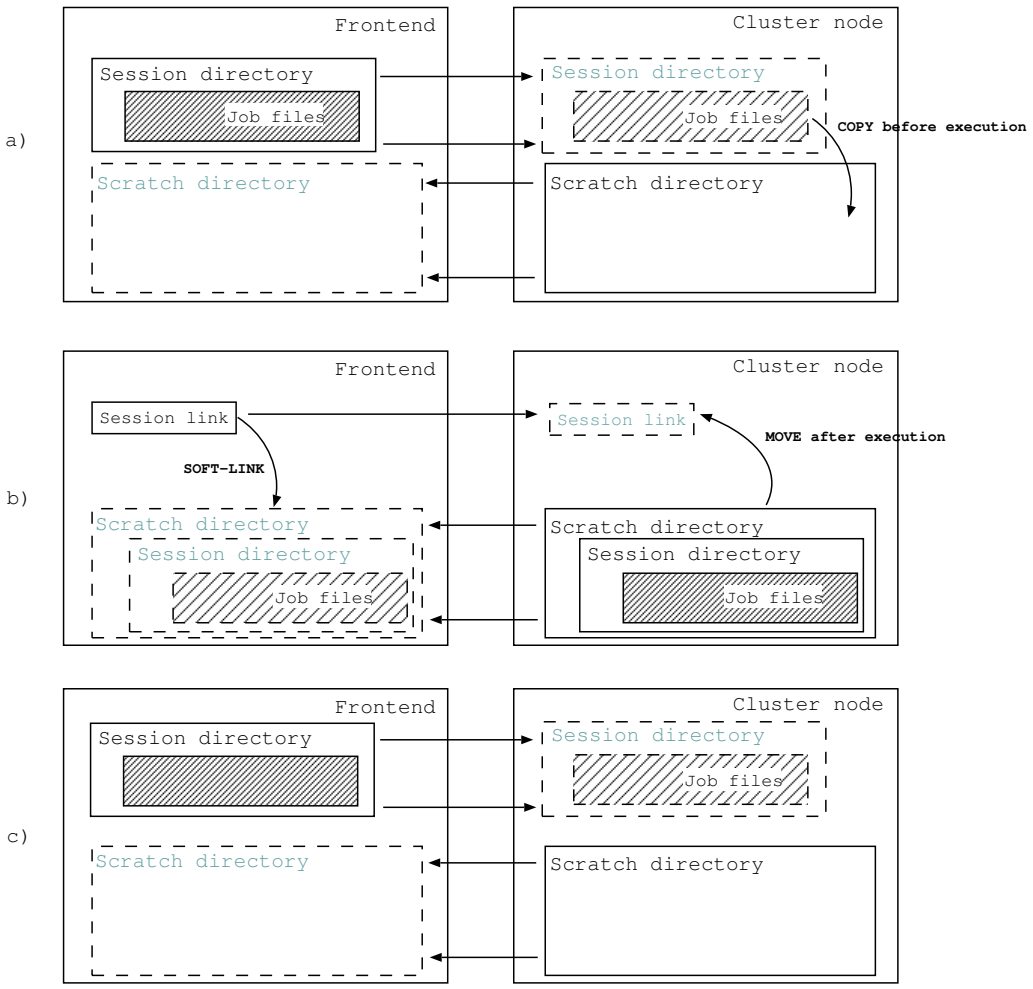


Figure 4: RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are set to values representing the scratch directory on the computing node and a way to access that scratch directory from the frontend respectively.

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_arg_# - command and arguments to be executed as specified in RSL.
- joboption_env_# - array of 'NAME=VALUE' environment variables (**not** bash array).
- 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" [14].

## 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" [14]. 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" [14]. 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 [12] 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* [15] and extensions to RSL [8] 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 *arcdata.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,
            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,
           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" [16].

# 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. | |

# References


[1] "The NorduGrid Collaboration," Web site. [Online]. Available: http://www.nordugrid.org

[2] M. Ellert *et al.*, "Advanced Resource Connector middleware for lightweight computational Grids," *Future Gener. Comput. Syst.*, vol. 23, no. 1, pp. 219–240, 2007.

[3] A. Konstantinov, *The HTTP(s,g) And SOAP Framework*, The NorduGrid Collaboration, NORDUGRID-TECH-9. [Online]. Available: http://www.nordugrid.org/documents/HTTP_SOAP.pdf

[4] W. Allcock *et al.*, "Data management and transfer in high-performance computational grid environments," *Parallel Comput.*, vol. 28, no. 5, pp. 749–771, 2002.

[5] B. Kónya, *The NorduGrid/ARC Information System*, The NorduGrid Collaboration, NORDUGRID-TECH-4. [Online]. Available: http://www.nordugrid.org/documents/arc_infosys.pdf

[6] "The Globus Resource Specification Language RSL v1.0." [Online]. Available: http://www-fp.globus.org/gram/rsl_spec1.html

[7] A. Anjomshoaa *et al.*, "Job Submission Description Language (JSDL) Specification, Version 1.0 (first errata update)," July 2008, GFD-R.136. [Online]. Available: http://www.gridforum.org/documents/GFD.136.pdf

[8] O. Smirnova, *Extended Resource Specification Language*, The NorduGrid Collaboration, NORDUGRID-MANUAL-4. [Online]. Available: http://www.nordugrid.org/documents/xrsl.pdf

[9] A. Konstantinov, *Protocols, Uniform Resource Locators (URL) and Extensions Supported in ARC*, The NorduGrid Collaboration, NORDUGRID-TECH-7. [Online]. Available: http://www.nordugrid.org/documents/URLs.pdf

[10] ——, *Configuration and Authorisation of ARC (NorduGrid) Services*, The NorduGrid Collaboration, NORDUGRID-TECH-6. [Online]. Available: http://www.nordugrid.org/documents/Config_Auth.pdf

[11] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997, available at: http://www.globus.org.

[12] A. McNab, "The GridSite Web/Grid security system: Research Articles," *Softw. Pract. Exper.*, vol. 35, no. 9, pp. 827–834, 2005.

[13] R. Alfieri *et al.*, "From gridmap-file to VOMS: managing authorization in a Grid environment," *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 549–558, 2005.

[14] B. Kónya, *NorduGrid ARC server installation instructions*, The NorduGrid Collaboration, NORDUGRID-MANUAL-2. [Online]. Available: http://www.nordugrid.org/documents/ng-server-install.html

[15] M. Ellert, *The NorduGrid toolkit user interface*, The NorduGrid Collaboration, NORDUGRID-MANUAL-1. [Online]. Available: http://www.nordugrid.org/documents/ui.pdf

[16] A. Konstantinov, *ARC::DataMove Reference Manual*, The NorduGrid Collaboration, NORDUGRID-TECH-8. [Online]. Available: http://www.nordugrid.org/documents/datamove.pdf