NORDUGRID-TECH-2

15/12/2006

THE NORDUGRID GRID MANAGER AND GRIDFTP SERVER

Description and Administrator's Manual

A.Konstantinov*

*aleks@fys.uio.no

Contents

1 Introduction

One of the problems the user of widely distributed computing networks faces is different configuration of *Computing Elements* (CE) controlled by different administrators. This makes even initial preparation of a job non-trivial task. This is especially important in 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 site-dependent pre- and post-computation job is necessary.

The aim of *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 it's connection to other parts please read "An Overview of The NorduGrid Architecture Proposal" [2]. It is **heavily using** Globus Toolkit^{TM} 2 as it's underlying software and currently **completely depends** on it.

Essential additionally part of the GM is the specialized GridFTP Server (GFS). This server supports rich enough subset of gsiftp protocol and has network and local file access parts separated. In context og GM it's main purpose is to provide control for job and access to the job files based on the user subject and job owner. Another option is Job Control Web Service (JCS) interface implemented as part of HTTPSD framework [3].

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

You should use this documment for advanced configuration purposes. It explains internals of the aforementioned tools and extended description of configuration options. For instalaltion and configuration refer to other documents avaialble 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 gets 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. GM does not guarantee the availability of any other places accessible by the job other than SD (unless such place is part of requested Runtime Environement). The SD is also the only place which is controlled by the GM. It is is accessible by the user from outside through 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 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 `nordugrid-cluster` under attribute `nordigrid-cluster-contactstring` and *jobid* (jobid corresponds to a SD).

Each job gets 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 GFS. All job parameters (not data) are passed to the GM through GFS in RSL [6] or JSDL-coded [7] description (job description - JD). The GM adds it's 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 file into the SD:

- Downloads initiated by the GM. Such files (name and source) are defined in the JD. It is a 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) and HTTPg (HTTP over GSI). Also some nonstandard sources are supported. Those are described below.
- 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, 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 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 GM).

Job stores output files in the SD. Those 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* like *Replica Catalog* (RC). The GM takes care of those files. They have to be specified in the JD. If job fails during any stage of processing no attept is taken to transfer those files to their final destination, unless there is option *preserve=yes* specified in their URLs.
- Files which are supposed to be fetched by the user. The user runs 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. Picture 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 UI or any other tool. Please remember that IS can manipulate with 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 GridFTP interface or to use query metod of JCS.

Configuration can put limits on amout of simultaneous jobs at some states. If such limit is reached job stays in it's current state waiting for free slot. This situation is presented by prepending current state name with **PENDING:** status mark.

Below is 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 limit on number of simultaneous **Preparing** jobs. Those jobs out of limit will stay in previous **Accepted** state with PENDING mark. Exceptions are jobs which has no files to be downloaded. Those are processed out of limits.

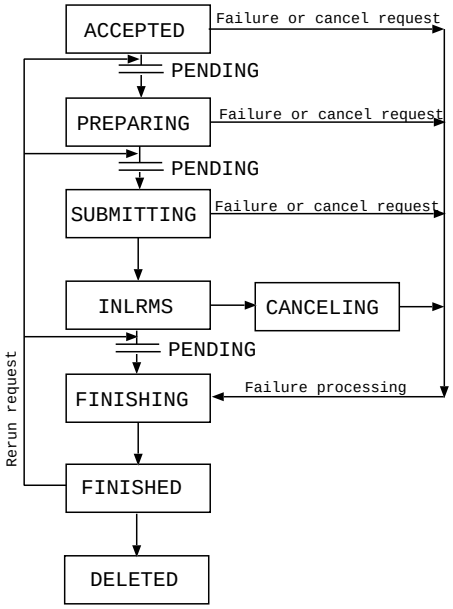

Figure 1: Job states

- **Submitting** - This is a point of interaction with *Local Resource Management System* (LRMS). At the moment PBS is supported best and correspoding backends are provided with default installation. 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 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 job finishes.

- **Cancelling** - Necessary action to cancel 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 at RC. The user can download data files from the SD by using 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 at NorduGrid Information System as attribute `nordugrid-pbs-job-sessiondirerasetime` of `nordugrid-pbs-job`. If job was moved to **Finished** because of failure, it may be restarted on request of client. Job is moved to a state previous to one which failed and is assigned mark PENDING. This is needed in order to not break the configuration limits. Exception is a job failed in **InLRMS** state and lacking input files specified in JD. Such job is treated like failed in **Preparing** state.

- **Deleted** - Job is moved to this state if user does not request job to be cleaned. Only minimal subset of information about such job is kept.

In the case of the 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 it's components support following data transfer protocols and corresponding URLs: *ftp, gsiftp, http, httpg, https, se, rc* and *rls.* For more information please see "Protocols, Uniform Resource Locators (URL) and extensions supported in ARC" document [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 configuration are exceeded.

- *job.ID.description* - contains the RSL description of the job.
- *job.ID.local* - information about 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 an URL or a file description. Example:

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

  url - ordinary URL for gsiftp, ftp, http, https or httpg protocols with the addition of '**replica catalog url**' (RC URL) and '**replica location service url**' (RLS 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. Those are not necessarily errors, but can be just useful information about actions taken during the job processing. In case of problem include content of that file while asking for help.
- *job.ID.diag* - information about resources used during execution of job and other information suitable for diagnostics and statistics. It's 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 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 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 implementation 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 unx ownership used by processes run with effective *user id* different form 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. It's interface can be found in Appendix B.

7 Cache

The GM can cache input files. Caching is enabled if corresponding command is present in configuration file. The GM does not cache files marked as executable in job. Caching can also be explicitly turned off by user for each file by using *cache=no* option in URL (for URL options read "Protocols, Uniform Resource Locators (URL) and extensions supported in ARC" [9]). The disc space occupied by cache is controlled by removing unused files. For more information look in section 8.2.

7.1 Structure

Cache directory contains plain files. Those are

- *list* - stores names of the files (8 digit numbers) and corresponding URLs delimited by blank space. Each pair is delimited by some amount of \0 codes. Also creation and expiration times are stored if available
- *old* - stores URLs which have been removed from cache. Records are delimited by some amount of \0 codes and are meant to be removed by some external routine.
- *new* - stores URLs which have been added to cache. Records are delimited by some amount of \0 codes and are removed when corresponding files are removed from cache. They can also be handled by some external routines. Every time record is added to *old* it is removed from *new*.
- *statistics* - consists of strings containing *name=value* . Following names are defined:
  - *hardsize* -size of file system for storing cached data
  - *hardfree* - amount of disc space available on that file system
  - *softsize* - if cache exceeds this size files are started being removed
  - *softfree* - space left till softsize (can be negative)
  - *claimed* - space used by files claimed by running jobs
  - *unclaimed* - space used by files not being currently used by any job
- *########.info* - stores state of file (######## stands for 8 digits). State is represented by one character:
  c - just created, content is empty.
  f - failed to download (treated same as 'c').
  r - ready to be used, content is valid.
  d - being downloaded. 'd' is followed by identifier of application/job downloading that file. During content's download this file has write lock set.
- *########.claim* - stores list of identifiers of applications/jobs using this file. Identifiers are stored one per line.
- ######## - files storing content of corresponding URL. These can be stored in separate directory.

Files *list, old , new* and *########.info* has to be stored on filesystem which has support for files' locking.

7.2 How it works

If job requests input file, which can/allowed to be cached, it is stored in cache directory instead and soft-link is created in the SD, pointing to that file. Or file can be stored in cache and then copied to the SD. Last option is more secure and hence advised.

Before downloading file the GM tries to determine it's size and to preallocate space in cache directory, by writing file of same size. If that fails (file system has no more space), it tries to remove oldest cache files, which are not being used by any job. That means **hard limit of cache size is space available at file-system**. In case cache gets full and it is impossible to free any space, download fails and is retried without using cache.

Before giving access to already cached file the GM contacts initial file source to check if user is allowed to do that if protocols allows to do that.

Also file creation or validity times are checked to make sure cached file is fresh enough. If it is impossible to obtain creation and invalidation times for file it is invalidated 24 hours after downloaded.

Also the GM checks cache periodically. If used space exceeds high water-mark given in configuration file (*softsize*) it tries to remove oldest unused files to reduce size to low water-mark. This sets soft limit of cache size.

There are 2 kinds of caches available. Files in *private* cache are owned by Unix user to which grid user is mapped. Those files are readable only by that particular Unix user. Another kind of cache is *shared*. Files are owned by Unix user who started GM and are readable by everyone.

8 Files and directories

8.1 Modules

The GM consists of 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 at the RC. It processes and updates the *job.ID.output* file.
- *cache-register* - Utility to register cached data into Indexing Services like RC and RLS. It reads and modifies cache informational files *old* and *new* 7. Configuration is read directly from the GM's configuration file 8.2. It is run by the GM every 5 minutes.
- *frontend-info-collector* - Utility to gather information about frontend and to put it into *job.ID.diag* file.
- *gm-kick* - Sends signal to the GM though FIFO file to wake it up. It's used to increase responsiveness of GM.

Following modules are always run under Unix account to which 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 and SGE. 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 the 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 less reliable *qstat* command for that. other backends use different techniques.

Also there is administrator utility:

- *gm-jobs* - prints list of jobs available on cluster and amount of jobs in every state.
  gm-jobs [-h] [-l] [-u uid] [-U name]
  -l-l - print more information about each job,
  -l-u - pretend utility is run by user with id *uid*,
  -l-l - pretend utility is run by user with name *name*.

GM comes with plugins useable for various authorization purposes (see for example description of *authplugin* command below):

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

8.2 Configuration of the Grid Manager

The GM configuration is done through single configuration file. Historically GM supports 2 kinds of configuration files. For old one it looks at following places:

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

And for new one in

- */etc/arc.conf*

The old configuration file consists of empty lines, lines containing comment (line starts from #) or configuration commands. Blank spaces in arguments must be escaped using '\' or arguments must be enclosed in '"'. Command line starts from command followed by arguments separated from command and between them by spaces.

The new configuration file can also contain empty lines and comments starting from #. It is separated into sections. Each sections starts from string containing

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

Each section continues till next section of end of file. One configuration file can have commands for multiple services/modules/programs. Each service get it's own section named after it. The GM uses section *[grid-manager]*. 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 format

- *name="arguments string"*.

Names are same as in old configuration file. The *argument string* consists of same arguments as in old format. And they must obey same rules.

Both files support almost same commands. Following commands are defined (examples are given for new format):

Global commads (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 go to 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 id if GM process will be stored. Defaults to *not write*.
- ***debug****=number* - specifies level of debug information. More information is printed for higher levels. Currently 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 (like 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 centralized service running at given *URL*. Multiple entries and multiple URLs are allowed. *number* specifies how long old records have to be kept if failed to be reported. That time is in days. 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 value specified in URL options.
- ***localtransfer****=yes|no* - specifies whether to pass file downloading/uploading task to computing node. If set to yes the GM won't download/upload files. Instead it composes script submitted to LRMS in way to make it do that. This requires installation of GM and Globus 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 - maximal amount of jobs being processed by GM. This does not limit amount of jobs, which can be suNOTE:bmitted to cluster
  max_running jobs - maximal amount 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 - maximal amount of jobs heavily using resources of frontend (applied before moving job to PREPARING and FINISHING states)
  emergency_frontend_jobs - if limit of *max_frontend_jobs* is used only by PREPARING or by FINISHING jobs aforementioned number of jobs can be moved to another state .This is used to avoid case then jobs can't finish due to big amount of recently submitted jobs.

max_transfered_files - maximal 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 for 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.

- ***cacheregistration****=yes|no* - enables or disables registration of cache data into Indexing Services like RC or RLS. The default is *no*. Only files dowmloaded through *meta-url* are registred. Registration is done to same service used for obtaining information about file. For this opeartion credentials of the GM (host key and certificate) are used. If required new files storage location is registered at Indexing Service with quasi-url *cache://hostname/* and name *hostname:cache* .

- ***authplugin****=state options plugin* - specifies *plugin* (external executable) to be run every time job is going to switch to *state*. Following states are allowed: ACCEPTED, PREPARING, SUBMIT, FINISHING, FINISHED and DELETED. If exit code is not 0 job is canceled by default. *Options* consist of *name=value* pairs separated by a comma. Following *name*s are supported:
timeout - specifies how long in seconds execution of the plugin 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 that 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 local identity of a user to which Grid identity is mapped in order to access filesystem at path specified in ***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/slow data fransfer is allowed to take place. Transfer is canceled if 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 be either URL or local path starting from '/'. It is advisable to end template with '/'.

- ***linkurl****=template replacement [node_path]* - mostly identical to *copyurl* but file won't be copied. Instead 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. Both *node_path* and *replacement* should not be URLs.

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

Per UNIX user commands:

- ***mail****=e-mail_address* - specifies an email address **from** which the notification mails are sent.
- ***defaulttl****=ttl [ttr]* - specifies the time in seconds for the SD to be available after job finished (*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. Queue name can also be specified in the JD (currently it is not allowed to override used LRMS by using JD). In new configuration file this command is called ***lrms***.
- ***session****=path* - specifies path to the directory in which the SD is created. If the path is * the default one is used - *$HOME/.jobs* . In new configuration file this command is called ***sessiondir***.
- ***cache****dir=path [link_path]* - specifies the directory to store cached data. Empty *path* disables caching. Default is not to cache data. Optional *link_path* specifies the path at which cache is accessible at computing nodes. If *link_path* is set to '.' files are not soft-linked, but copied to session directory. In old configuration file this command is called ***cache***.
- ***privatecache****=path [link_path]* - same as *cache* command, but cache belongs (owned) to user. For shared caches use 'cache'.
- ***cachedata****=path* - allows to specify separate place to store cache files containing data itself. This can be useful in case of big data storage available only on NSF server which does not support file locking. If command or *path* is missing - default is to store data at place specified in *cache* or *privatecache* command, together with control files.
- ***cachesize****=high_mark [low_mark]* - specifies high and low water-mark for space used by cache. Values are specified in bytes. Both *high_mark* and *low_mark* can be negative values. In that case corresponding positive value means space left on filesystem. If *low_mark* is omitted it becomes equal to *high_mark*. By default this feature is turned off. To turn it off explicitly *cachesize* without parameters should be specified. If turned off cache will grow up till it fills whole file system.
- ***maxrerun****=number* - specifies maximal number of times job will be allowed to rerun after it failed in LRMS. Default value is 2. This only specifies a upper limit. Actual number is provided in job description and defaults to 0.

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

- ***control****=path username [username [...]]* - This option initiates 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]).
  Also the special name '.'(dot) can be used. Corresponding control directory will be used for **any** user. This option should be the last one in the configuration file. In new configuration file command ***controldir****=path* is also available. It uses special username '.' and is always executed last independent of placement in file.

- ***helper****=username command [argument [argument [...]]]* - associates external program with the local UNIX user. This program will be kept running under account of the specified user. *username* stands for the name of the user. Special names can be used: '*' - all names from /etc/grid-security/grid-mapfile, '.' - root user. The user should be already configured with *control* option (except root, who is always configured). *command* is an executable and *argument*s are passed as arguments to it.

Following are global commands supported only in new configuration file. Most of them are specific to underlying LRMS (PBS in this case) and are passed in environment variables if 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 an access to session directory through a shared filesystem like NFS. Corresponds to an environement variable RUNTIME_NODE_SEES_FRONTEND.
- ***nodename****=command* - command to obtain hostname of computing node.
- ***scratchdir****=path* - path on computing node where to move session directory before execution.
- ***shared_scratch****=path* - path on frontend where ***scratchdir*** can be found.
- ***nodename****=command* - command to obtain hostname of computing node.

In the command arguments (paths, executables, ...) 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 - look command 'defaultlrms'

%L - default lrms - look command 'defaultlrms'

%W - installation path - ${NORDUGRID_LOCATION}

%G - globus path - ${GLOBUS_LOCATION}

%c - list of all control directories

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

%S - job's state (for *authplugin* plugins only, substituted in runtime)

%O - reason (for *localcred* plugins only, substituted in 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 - name log file (overwrites value in configuration file),
-P - name for file containing process id (overwrites value in configuration file),
-U - user and gropu id to use for running daemon,
-F - do not make process daemon,
-c - name od 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

Default location of the GFS configuration file is */etc/arc.conf* or *$NORDUGRID_LOCATION/etc/gridftpd.conf*. Format of these configuration files is similar to that of the GM. It also supports generic commands described at the beginning of 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 can't 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 agains 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 computer. You can write IP address directly instead of *hostname*. This command may be if server is situated behind 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 specified *rule*. If specified rule could not produce any mapping, next command is used. Mapping stops at first matched rule. Following rules are supported:

  - ***mapfile*** *file* - user's subject is matched against list of subjects stored inspecified file, one per line followed by local UNIX name.
  - ***simplepool*** *directory* - user is assigned one of 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 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 path to shared library of LCMAPS, either absolute or relative to *directory*; *directory* is path to LCMAPS installation directory, equivalent of LCMAPS_DIR variable; *database* is path to LCMAPS database, equivalent to LCMAPS_DB_FILE variable. Each arguments except *library* is optional and may be either skiped 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. Rule matches if exit code is 0 and there is UNIX name printed on *stdout*. Name may be optionaly followed by UNIX group separated by ':'. In arguments following substitions are applied before 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 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 mapping rule username is one, provided by rule. Otherwise specified *unixname:unixgroup* is taken. Both *unixname* and *unixgroup* may be either omited or set to '*' to specify missing value.
- ***groupcfg***=*name* - is put into subsections representnig plugin or [group] section and defines if that section is effective. In old format it selects the group to which all following lines apply. Only unaffected option is ***groupcfg***. If name is empty (or no groupcfg is used at all) following lines apply to all users.

Subsections of *gridftpd* section specifies plugins which serve virtual FTP path (similar to mount command of UNIX). Name of subsection is irrelevant. In old format this section starts with command ***plugin*** *path library_name* and ends with keyword ***end***. Inside subsection following commands are supported

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

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

  - *jobplugin.so* does not require any specific options in case of old configuration format. It reads the configuration file of the GM located at the standard place as specified in the section 8.2. Following options are supported:
    * ***configfile***=*path* - defines non-standard place for GM's configuration file,
    * ***allownew***=*yes*|*no* - specifies if new jobs can be submitted. Default is *yes*.
    * ***unixgroup***/***unixvo***/***unixmap*** - same options like in top-level GFS configuration. If mapping succeeds obtained local user will be used to run submitted job.
  - *filepligin.so* supports 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 the list of the following keywords:
  - ***nouser*** - do not use local file system rights, only use those specifies 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 specified 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 already 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* (second number is reserved for the future usage).
  - ***mkdir*** *owner:group permissions_or:permissions_and* - allow creating new directories.

– *gaclpligin.so* does not have options in case of the old configuration. First line of it's configuration contains local path (root directory) served by it. Rest till keyword ***end*** contains GACL [12] XML used to setup initial access rules for every newly created file and directory. If GACL XML is empty then there will be no default ACLs created for new files and directories. That means ACL of parent directory will be used.
  For the new configuration format following options are supported: ***gacl=****gacl* - GACL XML, ***mount****=path* - local path server by plugin.
  XML may contain variables which are replaced with values taken from client's credentials. 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 root directory must contain *.gacl* file with initial ACL. Otherwise rule will be "deny all for everyone".

Some configuration parameters can be specified from 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 - name log file (overwrites value in configuration file),

-*P* - name for file containing process id (overwrites value in configuration file),
-*U* - user and gropu id to use for running daemon,
-*F* - do not make process daemon,
-*c* - name od configuration file,
-*p* - TCP/IP port number,
-*n* - maximal number of simultaneously served connection,
-*b* - default size of buffer used for data transfer (default is 64kB),
-*B* - maximal size of buffer used for data transfer (default is 640kB).

8.4 Authorization

Authorization is performed at GFS by applying 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 as $NORDUGRID_LOCATION and following sub-directories are used:

$NORDUGRID_LOCATION/bin - tools
$NORDUGRID_LOCATION/libexec - program modules used by GM
$NORDUGRID_LOCATION/etc - configuration files, deprecated, central configuration file is used by deault
$NORDUGRID_LOCATION/sbin - daemons
$NORDUGRID_LOCATION/lib - gridftp server's plugins and API libraries
The GM also uses following directories:

- *session root directory* - In this directory the SD is created. It can be multiple directories for the various users specified in the configuration file.
  There are 2 processes which need to have a permissions to create new files and directories in it. Those are GM and GFS.
  If any of those processes processes are run under dedicated user account, that account need full permissions in the *session root directory*.
  If those processes are run under *root* account make sure *session root directory* is not on filesystem which limits capabilities of *root* user. For example NFS with *root_squash* option.
  If there is need to run processes under *root* account (to run jobs in LRMS under different users' accounts) but there is no way to provide suitable *session root directory* use *norootpower* command in configuration of the GM. In that case GM and GFS will use identity of local user to which Grid identity is mapped to access *session root directory*. Hence those users will need full access there.
  The GM creates SD with proper ownership and permissions for local identity used to run job. Some filesystems require *executable* permissions on *session root directory* to be set for local identity in order to access any file or subdirectory there.
  This directory should also be shared among cluster nodes in order for job to access input files. Or internal means of LRMS must be used to transfer files to executing node. For more see section 8.6.

- *control directory* - In this directory the SD stores an information about the accepted jobs. Both GM and GFS processes must have full permissions there.

Also subdirectory called *log* is 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. And this number is slowly growing. Features explained below are for **PBS** backend. This support is provided through *submit-pbs-job*, *cancel-pbs-job*, *scan-pbs-job* scripts. *submit-pbs-job* creates job's script and submits it to PBS. Created job's script is responsible for moving data between frontend machine and cluster node (if required) and execution of actual job. Alternatively it can download input files and upload output if *"localtransfer no"* is specified in the configuration file.

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

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

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

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

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

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

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

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

Figures 2,3,4 present some possible combinations for RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NOD and explain how data movement is performed. Pictures a) correspond to situation right after all input files are gathered in session directory and actions taken right after job's script starts. Pictures b) show how it looks while job is running and actions which are taken right after it finished. Pictures c) stand for final situation, when job files are ready to be uploaded to external storage element or be downloaded by user.

8.7 Runtime environment

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

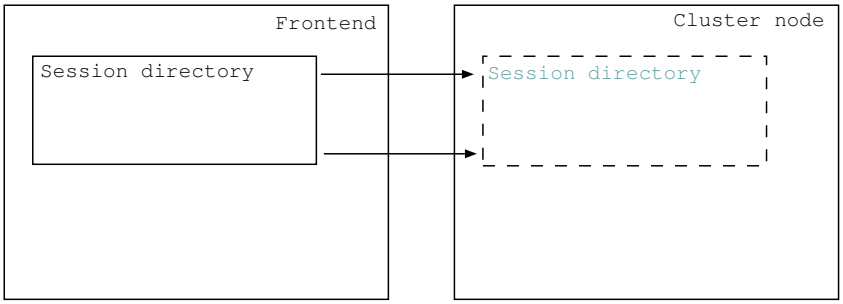

Figure 2: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE undefined. Job is executed in session directory placed on frontend.

- joboption_directory - session directory.
- joboption_args - command 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 beeing 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_maxcputime - amout of CPU time requested (minutes).
- joboption_maxmemory - amout 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 main executable. Or *joboption_runtime* could be expanded if current one depends on others.

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

With argument '2' scripts are executed after main executable finished. 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 computing node and is not reliable. If the job is killed by LRMS they most probably won't be executed.

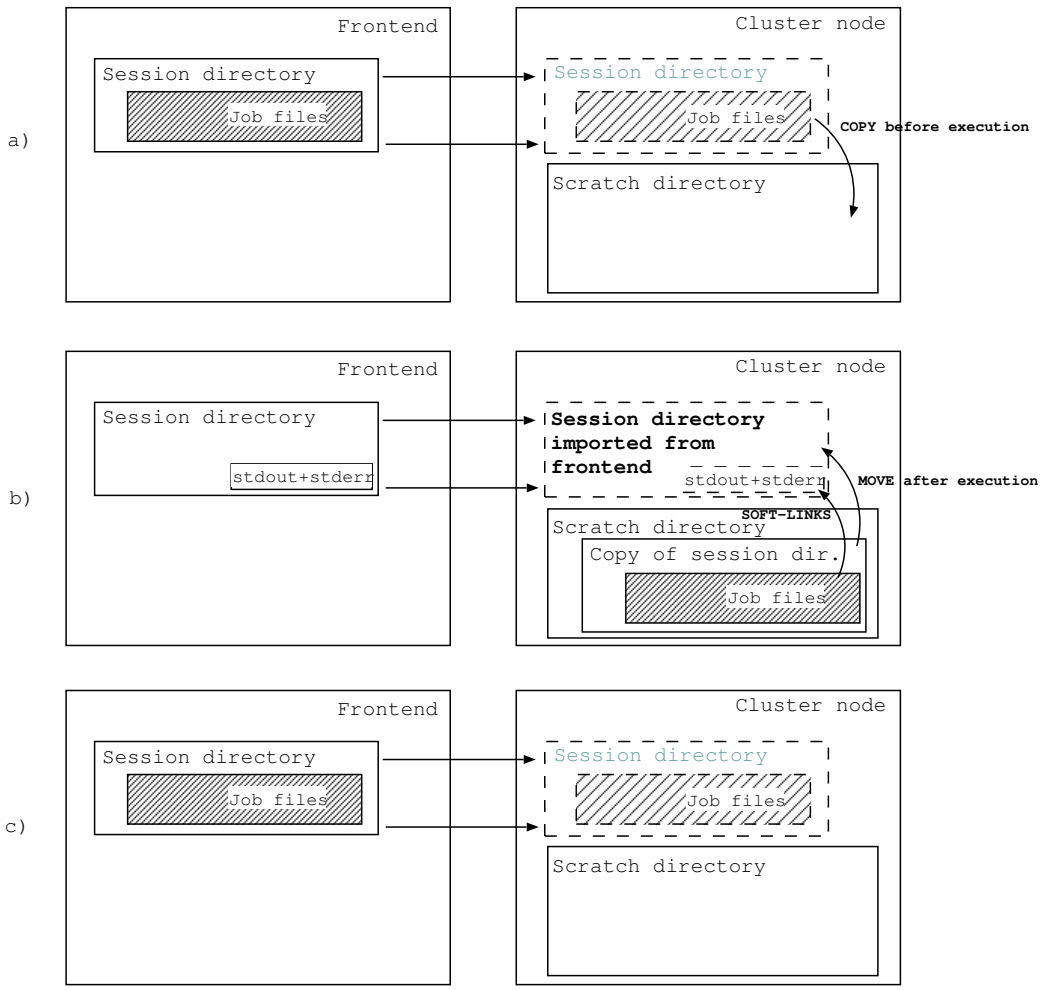

Figure 3: RUNTIME_LOCAL_SCRATCH_DIR is set to value representing sratch directory on computing node, RUNTIME_FRONTEND_SEES_NODE undefined.

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

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

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

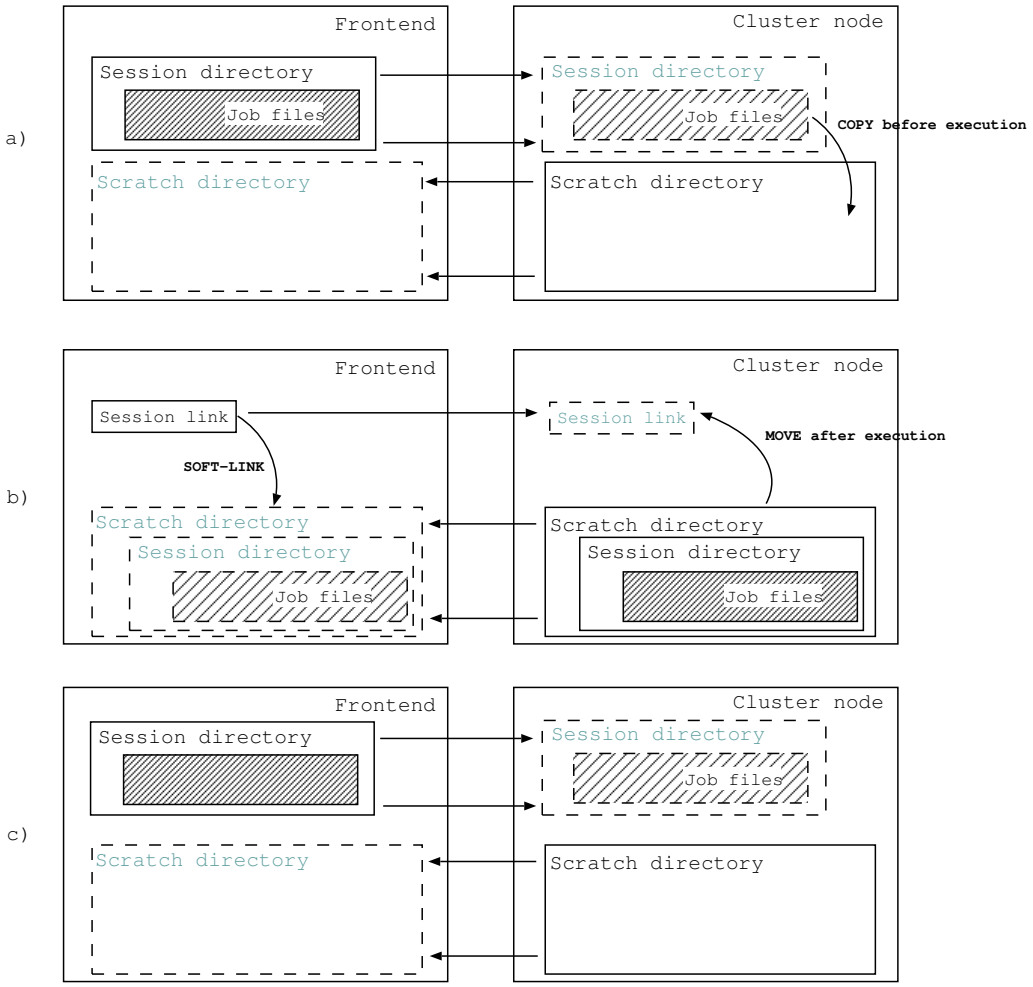

Figure 4: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are set to valuea representing sratch directory on computing node and way to access that scratch from frontend correspondingly.

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

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

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

9 Installation

To install GM as part of ARC-enabled site please read “NorduGrid ARC server installation instructions” at `http://www.nordugrid.org/documents/ng-server-install.html`.

9.1 Requirements

The GM is mostly written using C++. It was tested and should compile on recent enough *Linux* systems using *gcc* compiler and *GNU make* (gcc versions 2.95, 2.96, 3.2, 3.4 were tested). You will also need *Globus Toolkit*^{TM} of 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” at `http://www.nordugrid.org/documents/ng-server-install.html`. Follow that manual to install GM, 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. You can chose the name of the user by using corresponding command in configuration file. It is better run GM as root if You want to serve few users.

The GM writes debug information into a file /var/log/grid-manager.log by default. . Also 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 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 GM and related software please read “NorduGrid ARC server installation instructions” at `http://www.nordugrid.org/documents/ng-server-install.html`. Follow that manual to install GM, 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 uses for plain file access with the configuration senitive to user subject and is not necessary for setting a NorduGrid compatible site. The *gaclplugin.so* uses GACL (`http://www.gridpp.ac.uk/authz/gacl/`) to control access to local file system. The *jobplugin.so* is using information about jobs being controlled by GM and provides access to session directories of the jobs owned by 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 the ARC only one *jobplugin.so* needs to be configured. It is advisable to use the template configuration file. You can leave only part which configures *jobplugin.so* plugin.

9.4 Usage

Refer to the description of the *User Interface* part [14] and extensions to RSL [8] for using the GM.

9.5 Unix accounts

Bot GM and GFS are designed to be run by *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. Although this means some functionality loss described below.

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

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

As long as GFS with *jobplugin* is run under non-root account there is no mapping from Grid identity to local UNIX account taking place. All alowed Grid users are assigned server's account and are then processes by GM using same account. Only way to overcome this limitation is to run one GFS per local account with proper access control configured.

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

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

Appendix A. Job control over jobplugin.so

Virtual tree

Under mount point of jobplugin gridftp client can see directories representing job belonging to user, who started client. Directory per job. Directory names are same as jobs' identifiers. Those directories are directly connected to session directories of jobs and contain same files and subdirectories. Except if jobs session directory is moved to computing node. In that case directories only contain files with redirected stdout and strderr as specified in xRSL.

If job's xRSL has *gmlog* specified job's directory also contains virtual subdirectory with same name, which contains files with information about job as created by GM. The most important are 'errors' and 'status'. 'errors' contains stderr of separate modules run by GM in order to process job (downloader, uploader, job's submission to LRMS). 'status' contains one word representing state of job.

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

Submission

Each xRSL put into directory "new" is accepted as job's description. jobplugin parses it and client gets positive response if there are no errors in request.

Job gets identifier and directory with corresponding name appears. If job's description contains input files which should be delivered from client's machine, client must upload them to that directory under specified names.

Because each job gets identifier there should be a way for client to obtain it. For that prior to providing xRSL client sends command CWD to change current directory to "new". In this way job's identifier is reserved, new directory corresponding to that identifier is created and client is redirected to it (as specified in FTP protocol). Job's description put into "new" will get reserved identifier.

Actions

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

Currently supported actions are:

cancel to cancel job

clean to remove job from computing resource

renew to renew credentials delegated to job

restart to restart job after failure at some phases

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

Cancel

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

Clean

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

Renew

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

Appendix B. Library *libarcdata*

libarcdata is now part of *libngui* library. It's functions are declared in a header file *arcdata.h*. They correspond to ng* utilities meant for data handling - *arcacl*, *arccp*, *arcls*, *arcrm*, *arctransfer*. It consists of 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 =
void arccp (const std::string& source_url, const std::string& destination_url, bool secure = false,
void arcls(const std::string& dir_url, bool show_details = false, bool show_urls = false, int recu
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),
```

Additionally this library contains C++ classes used by *ng** data management utilities. Those are described in “ARC::DataMove Reference Manual”.

Appendix C. Error messages of GM

If job has not finished successfully the GM put one or more lines into *job.ID.failed*. Possible valuesinclude those generated 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.

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. [Online]. Available: http://www.nordugrid.org

[2] A. Wäänanen, "An Overview of an Architecture Proposal for a High Energy Physics Grid," in *Proc. of PARA 2002, LNCS 2367, p. 76*, J. Fagerholm, Ed. Springer-Verlag Berlin Heidelberg, 2002.

[3] A. Konstantinov, *The HTTP(s,g) And SOAP Framework*, The NorduGrid Collaboration, NORDUGRID-TECH-9.

[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.

[6] The Globus Resource Specification Language RSL v1.0. [Online]. Available: http://www-fp.globus.org/gram/rsl\protect\T1\textunderscorespec1.html

[7] A. Anjomshoaa *et al.* (2005, December) Job submission description language (jsdl) specification v1.0. GFD-R-P.056. [Online]. Available: http://www.ggf.org/ggf_docs_final.htm

[8] O. Smirnova, *Extended Resource Specification Language*, The NorduGrid Collaboration, NORDUGRID-MANUAL-4.

[9] A. Konstantinov, *Protocols, Uniform Resource Locators (URL) and Extensions Supported in ARC*, The NorduGrid Collaboration, NORDUGRID-TECH-7.

[10] ----, *Configuration and Authorisation of ARC (NorduGrid) Services*, The NorduGrid Collaboration, NORDUGRID-TECH-6.

[11] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.

[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] M. Ellert, *The NorduGrid toolkit user interface*, The NorduGrid Collaboration, NORDUGRID-MANUAL-1.