Nordic Testbed for Wide Area Computing And Data Handling

4/11/2003

THE NORDUGRID GRID MANAGER AND GRIDFTP SERVER

Description and Administrator's Manual[*]

A.Konstantinov

[*]Comments to: 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 and transfer or store output results.

The GM is part of the NorduGrid software and for it's connection to other parts "An Overview of The NorduGrid Architecture Proposal" [2] should be studied. It is **heavily using** Globus Toolkit^{TM} as it's underlying software and **completely depends** on it.

Additionally part of the GM, the specialized GridFTP Server (GFS) is installed. This server supports gsiftp protocol (reach enough subset) and has network and local file access parts separated. It's main purpose is to provide access to the user files based on the user subject and job owner.

You should use this manual for installation and configuration purposes only if You are installing the GM separately from NorduGrid Toolkit. Else treat it as document explaining internals of the aforementioned tools and for instalaltion and configuration refer to `http://www.nordugrid.org`. And hence skip section 9.

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 SD. The job is supposed to produce new data files also in SD. GM does not guarantee the availability of any other places accessible by the job other than SD. 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 exchange of data (including also program modules) is done through GridFTP protocol [3] **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 a handle which identifies the job in the GM and the NorduGrid Information System [4].

Each job is initiated and controlled through GFS. Initially job control through Globus GRAM [5] was supported. Starting from Globus version 2.2 GRAM is not supported by GM anymore due to changed internal interface. All job parameters (not data) are passed to the GM through GFS in RSL-coded [6] description (job description - JD). The GM adds it's own attributes to Globus RSL [7].

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 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: gsiftp and ftp (http and https should work too, but were not tested).

- 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.

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 a *Replica Catalog* (RC). The GM takes care of those files. They have to be specified in the JD.
- 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 using the UI or any other

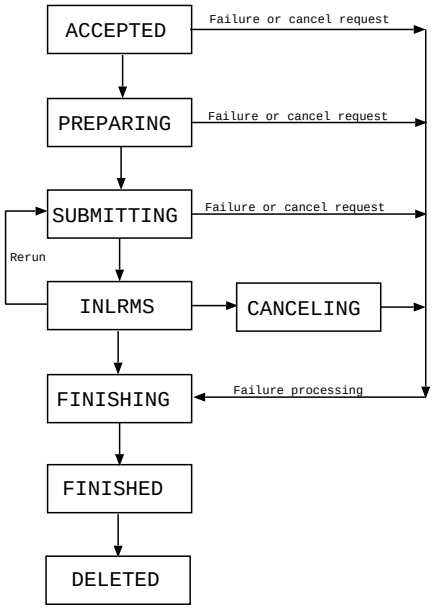

Figure 1: Job states

tool. 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 move 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.

- **Submitting** - This is a point of interaction with *Local Resource Management System* (LRMS). At the moment only PBS is supported. 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**.
- **InLRMS** - The job is queued or being executed in the LRMS. The GM takes no actions except waiting until job finishes.
- **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.
- **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). The 'deletion' time can be queried at NorduGrid Information System as attribute `nordugrid-pbs-job-sessiondirerasetime` of `nordugrid-pbs-job`.
- **Deleted** - Job is moved to this state if user does not request job to be cleaned.

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.

If the job is allowed to rerun it can go into a loop between **InLRMS** and **Submitting**. However, the maximum number of times this can happen can be specified in the GM configuration or in the JD.

5 URLs

The GM and it's components support following data transfer protocols and corresponding URLs:

- *ftp* - ordinary FTP
- *gsiftp* - GridFTP, enhanced FTP protocol with security, encryption, etc.
- *http* - ordinary HTTP with PUT and GET methods
- *https* - HTTP wrapped with Globus GSI

In addition to standard URL fields GM supports *options.* Options are of kind *name* or *name=value* and are inserted into URL after host and port and are separated by ';'

protocol://host[:port][;option[;option[...]]]/path

Following options are supported:

threads=# specify number of parallel transfers to use (currently works for GridFTP only, default is one),

cache=yes|no whether GM should cache file (default is yes),

secure=yes|no whether data should be transfered using encryption (currently works for GridFTP only, default is no unless specified differently in configuration),

exec=yes|no means file should be treated as executable (currently only used internally).

Example: *gsiftp://grid.domain.org:2811;threads=10;secure=yes/dir/input_12378.dat*

Also following meta-data servers are supported:

- *rc* - Globus Replica Catalog
- *rls* - Globus Replica Location Service

URLs for source/destination files specified through meta-data servers look like

- *rc://[location[|location[...]]][@host[:port]/distinguished_name]/lfn*
- *rls://[url[|url[...]]@]host[:port]/lfn*

Here

lfn Logical File Name, that should be used for registering/querying in the RC,

distinguished_name DN of collection in LDAP server used for registering/querying,

location name of location (usually this is host[:port] of SE), each *location* can have *options* appended to it in a way *location*;*options*.If only *;options* part is preset, those options are treated as specified for every location.

url full or partial URL corresponding to physical location of file (partial URLs are allowed only if file is going to be retrieved).

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.

- *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 time when the job was accepted
  - *lifetime* - time to live for the SD after job finished
  - *cleanuptime* - time when job will to removed from cluster and SD deleted
  - *notify* - email addresses and flags to send mail to about job specified status changes
  - *processtime* - when to start processing the job
  - *exectime* - when to start job execution
  - *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)
  - *sessiondir* - SD of job
  - *failedstate* - state at which job failed (available only if it is possible to restart job)

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 or https 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. It's format is similar to that of *job.ID.local*. The following names are defined:
  - *nodename* - name of computing node which was used to execute job,
  - *runtimeenvironments* - used runtime environments separated by ';',
  - *exitcode* - numerical exit code of job,

  and other information provided by GNU *time* utility

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 are a libraries *libui* and *libngdata* distributed as part of the GM. *libui* provides functionality to submit job, retrieve results and renew credentials and used by the UI. Interface is described in Appendix B. *libngdata* (available only if built using autotools) provides support for moving data between different URLs. It's interface can be found in Appendix C.

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 look section 6). The disc space occupied by cache can be 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. The records 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. The records are removed when corresponding files are removed from cache. They can also be handled by some external routines.
- *statistics* - stored 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
- *new* - list of URLs added to cache. Separated by \0.
- *old* - list of URLs removed from cache. Separated by \0. Every time record is added to *old* it is removed from *new*.
- *########.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 uses, 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 file 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. Alternatively file can be stored in cache and then copied to the SD.

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, job fails.

Before giving access to cached file the GM contacts initial file source to check if user is allowed 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. This is the only module which **can** be run from root account. Other modules will be always run by *grid-manager* using account of the owner of the job.
- *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.
- *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. The only supported LRMS is PBS (and the name is *submit-pbs-job*). This module is responsible for the job submission to the LRMS. This is a shell script derived from corresponding file of Globus Toolkit*TM*.
- *cancel-*-job* - This one is for canceling the job, which was submitted to LRMS.
- *parse-*-log* - This shell script is responsible for notifying the GM about completion of the job.
- *scan-*-job* - Alternative to *parse-*-log*. In case of PBS *parse-pbs-log* uses logs to find out status if a job. While *scan-pbs-job* uses unreliable *qstat* command for that.
- *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.

Also available few administrator and user level utilities

- *ngcopy* - copy file from *url* to *url*. Accepts both ordinary and RC URLs. Syntax:
  ngcopy [-h] [-v] [-c cache_path [-C cache_data_path]] [-d level] source destination
  -c - use cache at 'cache_path'
  -C - store cached data at 'cache_data_path'
  -s - use secure data transfer (this eats a lot of CPU power).
  Source URL can end with '/'. In that case whole fileset (directory) will be copied. Also if destinations end with '/' it is extended with part of source URL after last '/'.
- *ngremove* - remove file at given *url*. Accepts both ordinary and RC URLs. In case if RC URL is given without location, deletes also meta-information about file (alsp known as logical file name).
  ngremove [-h] [-v] [-c] [-d level] url
  -c - continue with meta-data even if it failed to delete real file.

- *ngls* - list contents of directory or file attributes acorresponding to provided URL. Accepts both ordinary and RC URLs. For RC URL currently LFN is ignored.
  ngls [-h] [-v] [-d level] [-l] [-L] url
  -l - print files' attributes
  -L - print files' attributtes and URLs which can be used to obtain that file.

- *ngacl* - manipulate ACL of file or directories on gridftp servers supplied with GACL access contorl. It either prints contents of GACL ACL XML on stdout or reads it from stdin. If UR: end with '/' directory ACL is read/written. Otherwise ACL of file is processed.
  ngacl [-h] [-v] [-d level] get|set url
  get|set - command to get or set file or directory ACL correspondingly.

  – common options are:
    -h - print short reminder
    -v - print version
    -d - set debug level

- *gm-jobs* - prints list of jobs available on cluster and amount of jobs in every state.
  gm-jobs [-l]
  -l-l - print more information about each job.

8.2 Configuration file of the Grid Manager

The GM configuration is done through single configuration file. The GM looks for the configuration file at the following places:

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

The configuration file consists of lines containing comment (line starts from #) or configuration options. Following options are defined:
Global options:

- ***joblog*** *[path]* - specifies where to store log file containing information about started and finished jobs.
- ***jobreport*** *[URL]* - specifies that GM has to report information about jobs being processed (started, finished) to centralized service running at given *URL*.
- ***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 submitted to cluster
max_running jobs - maximal amount of jobs passed to Local Resource Management System
Missing value or -1 means no limit.

- ***maxlod*** *[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.

- ***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 timeput plugin* - specifies *plugin* (external executable) to be run every time job is going to switch to *state*. Execution of *plugin* may not last longer than *timeout* seconds. Following states are allowed: ACCEPTED, PREPARING, SUBMIT, FINISHING, FINISHED and DELETED.

- ***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 options:

- ***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*.

- ***defaultlrms*** *default_lrms_name default_queue_name* - specifies default names for the LRMS and queue, which are used if not specified in the JD (currently it is not allowed to override used LRMS by using JD).

- ***session*** *path* - specifies path to the directory in which the SD is created. If the path is * the default one is used - *$HOME/.jobs* .

- ***cache*** *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.
- ***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 [8]).
  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.
- ***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 executables and *argument*s are passed as arguments to it. At the moment this option is supposed to be used to run *parse-*-log* programs. If all the users are supposed to use the same LRMS (the only option supported now) and job control directory is the same it is easier to have one helper process running as root. *parse-*-log* is designed in the way it serves all the users if run by root.

8.3 Configuration file of the GridFTP Server

The GFS configuration file is at *$NORDUGRID_LOCATION/etc/gridftp-server.conf* . Format of this configuration file is similar to that of the GM.

- ***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.

- ***group*** *name* - define the group containing the user with the specified subjects. Each line contains rule use to define if connected user belongs to that group. For available rules and their description read below at 8.4. Keyword ***end*** is used to mark end of definition.
- ***unixmap*** *unixname[:unixgroup] rule* - define local Unix user and optionally group used to represent connected client. *rule* is the same as for ***group*** command. If there is no suitable mapping for user, default is to take one from grid-mapfile.
- ***groupcfg*** *name* - select 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.
- ***plugin*** *path library_name* - make plugin *library_name* to serve virtual path *path* (similar mount command of Unix). Following lines contain plugin specific options till keyword ***end***. GFS comes with 3 plugins: *filepligin.so, gaclplugin.so* and *jobplugin.so*.
  - *jobplugin.so* does not have any specific options, so the following line should contain only word *end*. It reads the configuration file of the GM located at the standard place as specified in the section 8.2.
  - *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. First line of it's configuration contains local path (root directory) served by it. Rest till keyword ***end*** contains GACL [9] 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. XML can 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[10] 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".

8.4 Authorization

Authorization is performed by applying set of rules. Each rule takes one line in the configuration file and can be preceded by modifiers - [+|-][!]

+ accept user if matches following rule (positive match, default action),

- reject use if matches following rule (negative match),

! invert matching. Match is treated as non-match. Non-match is treated as match, either positive ("+" or nothing) or negative ("-").

Processing of rules stops after first positive or negative match is reached.
Following rules are supported:

- [***subject***] *subject* [*subject* [...]]- accept user with one of specified subjects
- ***file*** [*filename* [...]] - read rules from specified files
- ***vo*** [*ldap://host:port/dn* [...]] - accept users listen in one of specified LDAP directories
- ***voms*** *vo group role capabilities* - accept user with VOMS proxy belonging to specified *vo* and *group* and having specified *role* and *capabilities*. '*' can be used to accept any value.
- ***group*** [*groupname* [...]] - accept user already belonging to one of specified groups.
- ***all*** - accept any user

Blank spaces in arguments must be escaped using '\' or arguments must be enclosed in '"'.

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 - program modules
$NORDUGRID_LOCATION/etc - configuration file
$NORDUGRID_LOCATION/sbin - System V start-up scripts
$NORDUGRID_LOCATION/lib - gridftp server's plugins
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.
  If You are using job submission through gridftp interface the session root directory and have You daemons run by root account You **do not need** to make it writebale for users, but they still need at least executable (x) access on it in order to .
  If You want to use Globus jobmanager note it is run under the user account. Hence administrator installing the GM **must** take care that session root directory is writable by the user, who is going to have SD there. Only Globus jobmanager pior and including version 2.0 is supported.
  This directory should also be shared among cluster nodes in order for job to access input files.

- *control directory* - In this directory the SD stores an information about the accepted jobs. Permission requirements are the same as those for the session root directory, except there is no need to keep executable permissions for all users.
  Also subdirectory called *log* is created there. It is used to store

8.6 LRMS support

The GM only supports PBS at the moment. This support is provided through *submit-pbs-job*, *cancel-pbs-job*, *scan-pbs-job* and *parse-pbs-log* 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.

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.

PBS_BIN_PATH - path to PBS executables.

TMP_DIR - path to directory to store temporary files.

RUNTIME_CONFIG_DIR - path where runtime setup scripts can be found.

GNU_TIME - path to GNU time utility.

NODENAME - command to obtain name of cluster's node.

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.

RUNTIME_FRONTEND_SEES_NODE - if defined should contain path corresponding to *RUNTIME_LOCAL_SCRATCH_DI* as seen on **frontend** machine.

Figures 2,3,4 present possible combinations for RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_N 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 and before executing job's main executable. Those scripts are requested by user through *runtimeenvironment* attribute in RSL and are run with only argument equal to '0' or '1' for creation of job's script and execution of job accordingly. In case of '0' argument some environment variables are defined and can be changed to influence job's execution later:

- joboption_directory - session directory.

- joboption_args - command to be executed as specified in RSL.

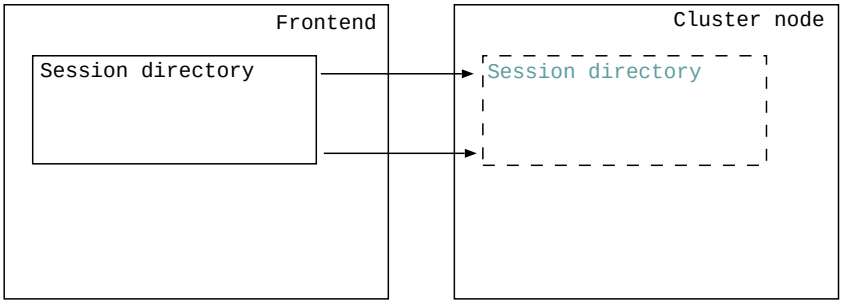

Figure 2: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE undefined. Job is executed in session directory placed on frontend.

- 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.

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

In case of '1' argument script is called just before optional staging to computing node is performed (described in section 8.6) and job is run. It is executed on computing node. It could for example adjust ***RUNTIME_LOCAL_SCRATCH_DIR*** and ***RUNTIME_FRONTEND_SEES_NODE*** variables and perform other necessary tasks to prepare environment for some third-party software package.

9 Installation

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 were tested). You will also need *Globus Toolkit*^{TM} *2.x* installed `http://www-unix.globus.org/toolkit/`.

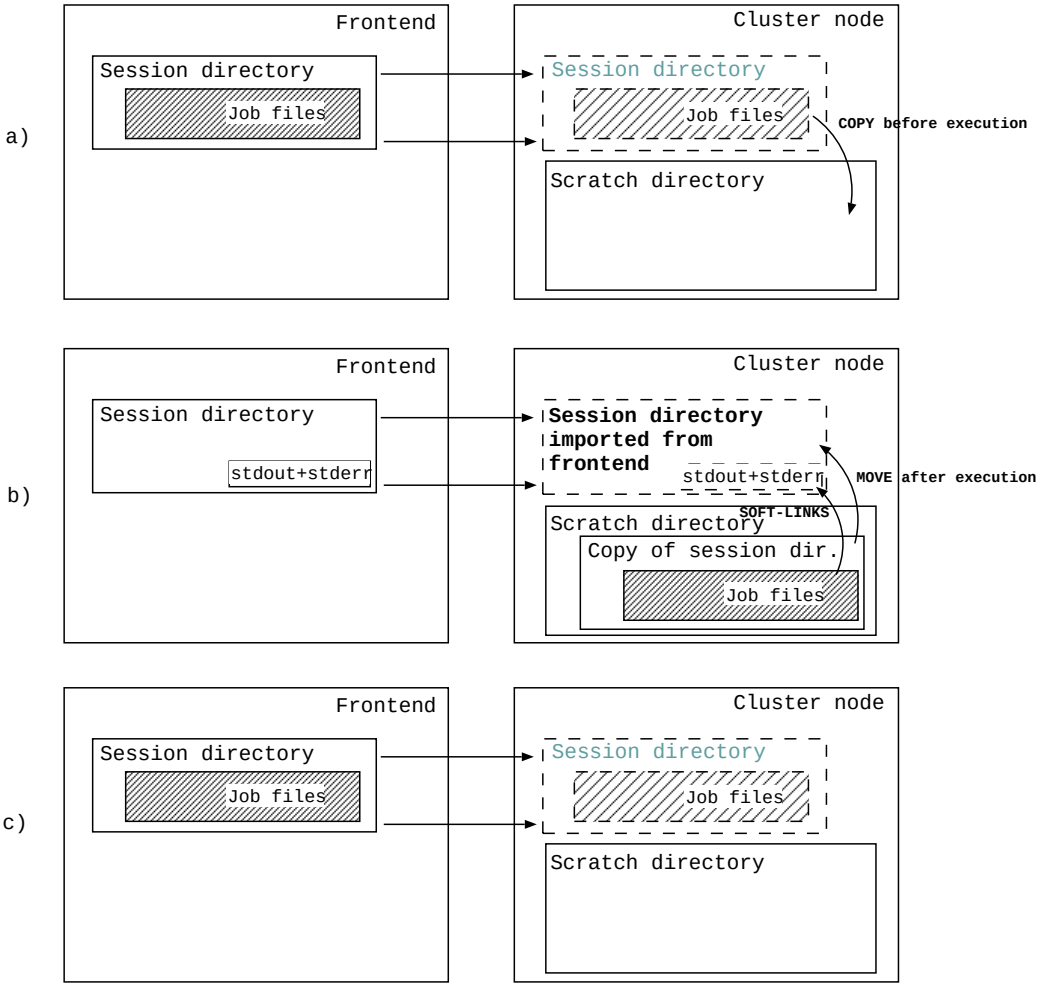

Figure 3: RUNTIME_LOCAL_SCRATCH_DIR is set to value representing sratch directory on computing node, RUNTIME_FRONTEND_SEES_NODE undefined.

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.

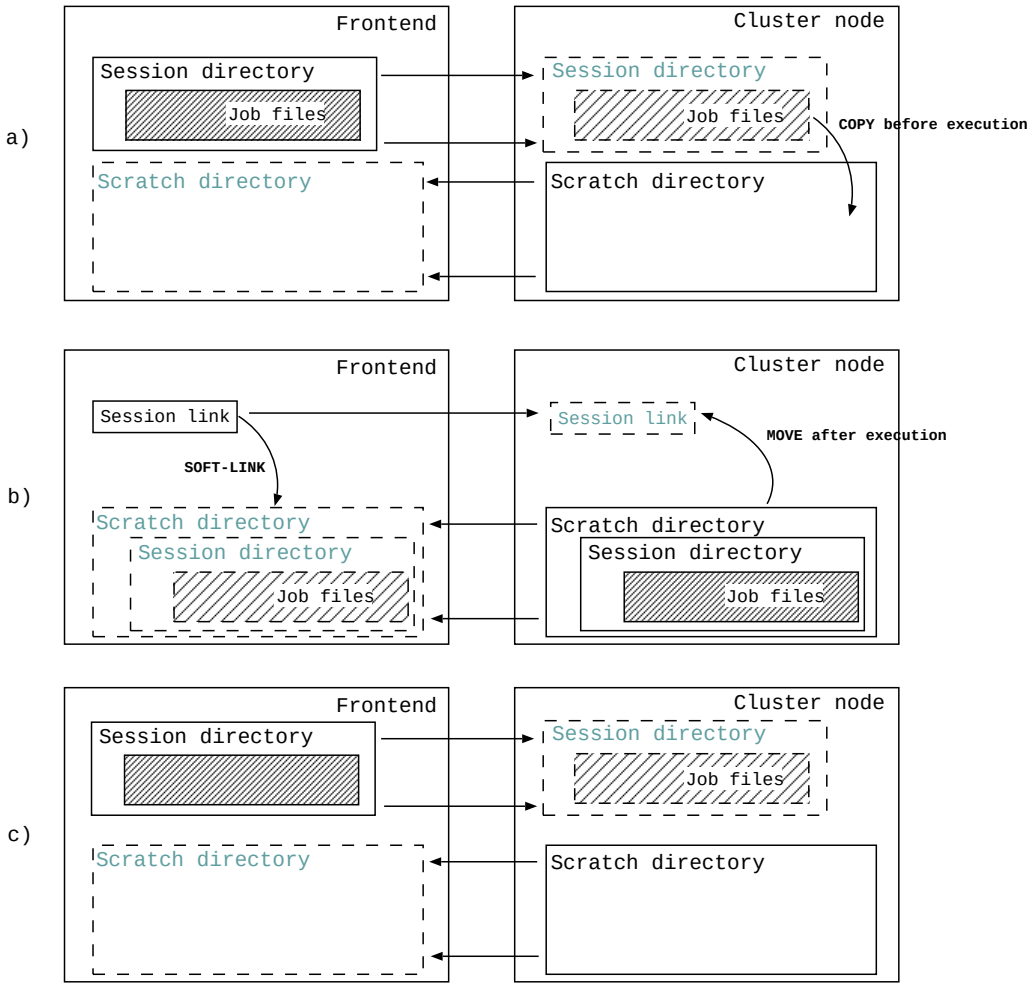

Figure 4: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are set to valuea representing sratch directory on computing node and way to access that scratch from frontend correspondingly.

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.

9.2 Obtaining

Get distribution of GM as part of NorduGrid Toolkit at `http://www.nordugrid.org/download/`. Pick the latest version. Download and unpack it.

To compile GM **only** go to directory grid-manager. **Read and edit** file Make.inc . Make sure GLOBUS_LOCATION points to the Globus installation directory and GLOBUS_FLAVOR is the one You have, *gcc32dbgpthr* or *gcc32pthr* are advised. The GM was tested only with *gcc32dgpthr* threaded version of Globus and it uses threads itself. So it most probably won't work properly with non-threaded version of Globus libraries. Variable NORDUGRID_LOCATION should contain path where the GM is to be installed. Make sure linker can find Globus libraries (use LD_LIBRARY_PATH environment variable for example).

Do not forget to edit variables which set the paths to the PBS installation: PBS_LOCATION and PBS_SPOOL .

Read comments to find out meaning of other variables.

9.3 Compilation

Run 'make' in the main source directory. This will create few executables in various sub-directories. Those are:

- grid-manager
- downloader
- uploader
- rsl/ng-parse-rsl
- misc/smtp-send
- globus-script-ng-submit
- init/grid-manager
- init/gridftp-server
- PBS/submit-pbs-job
- PBS/cancel-pbs-job
- PBS/parse-pbs-log
- PBS/scan-pbs-job
- gridftp/gridftp-server

Few libraries will also be created:

- libui.a
- gridftp/fileplugin/fileplugin.so
- gridftp/jobplugin/jobplugin.so
- gridftp/gaclplugin/gaclplugin.so

9.4 Installation

Run 'make install' in the main source directory. This will create directories

$NORDUGRID_LOCATION/bin
$NORDUGRID_LOCATION/sbin
$NORDUGRID_LOCATION/etc
$NORDUGRID_LOCATION/lib
$NORDUGRID_LOCATION/libexec
$NORDUGRID_LOCATION/include

and install few files there.

9.5 Configuration of the GridManager

To make GM to **interoperate with other parts** of the NorduGrid software it should exist **only one** session root directory and **only one** control directory. It is advisable to use the template configuration file $NORDUGRID_LOCATION/etc/grid-manager.conf.template . Copy it to $NORDUGRID_LOCATION/etc/grid-manager.conf. Then read section 8.2 and comments inside configuration file and edit it if needed.

Now place a file $NORDUGRID_LOCATION/sbin/grid-manager into /etc/rc.d/init.d/ and enable it with *chkconfig*. Alternatively You can make a soft-link. This is SystemV style start-up script. If You have BSD style system configuration You can call it from /etc/rc.d/rc.local. In the other cases read Your system's manual. 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 modifying variable GM_USER in start-up script. It is better to keep it empty and run GM as root if You want to serve few users.

You may need to adjust few paths in files $NORDUGRID_LOCATION/bin/submit-pbs-job and $NORDUGRID_LOCATION/bin/cancel-pbs-job . You can edit variables described in 8.6 or set them in environemnt before starting GM.

Unless You want to use GFS for job submission (strongly advised) now it's time to configure Globus job-manager. Please note, that the GM does **not** support submission through Globus GRAM (gatekeeper, job-manager) for version newer than 2.0. Few files called

globus-script-ng-submit
globus-script-ng-queue
globus-script-ng-rm
globus-script-ng-poll
ng-parse-rsl

were installed in Your $GLOBUS_LOCATION/libexec . You have to to add new resource to Globus gatekeeper configuration with '-rdn ng'. You can choose any name for it, but it is advisable to call it *jobmanager-ng*. For how to do that study Globus distribution documentation. Look for it at `http://www-unix.globus.org/toolkit/`.

9.6 Configuration of the GridFTP Server

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 NorduGrid software only one jobplugin.so is required to be configured. It is advisable to use the template configuration file $NORDUGRID_LOCATION/etc/gridftp-server.conf.template . Copy it to $NORDUGRID_LOCATION/etc/grid-manager.conf. Then read section 8.3 and comments inside configuration file and edit it if needed. You can leave only part which configures *jobplugin.so* plugin.

There is no additional configuration job required for the GFS.

9.7 Running

To start the GM run the System V start-up script $NORDUGRID_LOCATION/sbin/grid-manager with an argument 'start'. To start the GFS run the start-up script $NORDUGRID_LOCATION/sbin/gridftp-server with an argument 'start'. Or if You added them to system configuration, behave according to Your systems requirements.

Both scripts also support other usual options like start, restart, etc. *grid-manager* script also accepts additional options:

lightcleanstart - after the GM starts it removes all jobs with states FINISHED,

cleanstart - all recognized jobs are removed,

distcleanstart - all files present in control and session directories are removed.

The GM writes debug information into *stderr* and startup script redirects it into a file /var/log/grid-manager.log. GFS startup scripts redirects server's output to /var/log/gridftp-server.log . Also file /var/log/gm-jobs.log (default path in configuration template) contains information about all started and finished jobs, 2 lines per job (1 when job is started and 1 after it finished).

9.8 Using

Refer to the description of the *User Interface* part [11] and extensions to RSL [7] for using the GM.

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. Directories 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 contains files with redirected stdout and strderr as specified in xRSL.

If job's xRSL has *gmlog* specified job's directory also contains 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 one additional directory named "new".

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.

Cancellation

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.

Cleaning

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".

Credentials renew

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

Appendix B. Library *libui*

Following interface is defined:

```
typedef enum {
  RSL_ACTION_REQUEST=0,
  RSL_ACTION_CANCEL =1,
  RSL_ACTION_CLEAN =2,
  RSL_ACTION_RENEW =3
} rsl_action;
int ui_downloader(const char* url,bool resursive,const char* path,const vector<string> &filenames,bool do
int ui_uploader(const char* resource,const char* rsl,char** job_id,const char* session_url,rsl_action act
```

ui_downloader is used to retrieve result of job and *ui_uploader* to submit job and to control it depending on value of *act*. Arguments are:

url,*session_url* - URL used to access the job,

path - local path used to store downloaded files,

filenames - names of files available at GM or those to upload to GM,

download_files - if true download files, otherwise just fill list of available files

remove_files - if true remove all files and job from GM,

debug - debug level,

resource - URL of resource to submit job to (usually this is just gsiftp URL of the root of SDs),

rsl_action - action to be performed:

RSL_ACTION_REQUEST - submit the job,

RSL_ACTION_CANCEL - cancel job being processed,

RSL_ACTION_CLEAN - inform GM it can remove job,

RSL_ACTION_RENEW - renew job's credentials.

Appendix C. Library *libngdata*

This interface simply contains entry points to *main* functions of the utilities provided with the GM. It consists of following functions:

```
int ngcopy(int argc,char** argv);
int ngremove(int argc,char** argv);
int ngacl(int argc,char** argv);
int ngls(int argc,char** argv);
```

References

[1] NorduGrid project. http://www.nordugrid.org

[2] An Overview of The NorduGrid Architecture Proposal. http://www.nordugrid.org/documents/nordarch.pdf

[3] GridFTP: Universal Data Transfer for the Grid. http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf

[4] The NorduGrid Information System. http://www.nordugrid.org/documents/ng-infosys.pdf

[5] Globus Resource Allocation Manager. http://www.globus.org/gram/

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

[7] Extended Resource Specification Language. http://www.nordugrid.org/documents/xrsl.pdf

[8] The Globus Project. http://www.globus.org/

[9] GACL - a Grid ACL manipulation library. http://www.gridpp.ac.uk/authz/gacl/

[10] VOMS Architecture (v1.1) (draft). http://grid-auth.infn.it/docs/VOMS-v1_1.pdfhttp://grid-auth.infn.it/docs/VOMS-v1_1.pdf, Authorization Working Group http://grid-auth.infn.it/.

[11] The NorduGrid User Interface.http://www.nordugrid.org/documents/NorduGrid-UI.pdf