NorduGrid ARC Tutorial Exercises

Juha Lento <juha.lento@csc.fi>
Arto Teräs <arto.teras@csc.fi>

CSC, Espoo, Finland
September 20, 2005

Contents

1 Introduction

This document contains examples and exercises for the NorduGrid ARC middleware tutorial.

A printed copy of the NorduGrid/ARC User Guide should be used on the side with this document. It is usually provided in the tutorial sessions, but it is also available on the web at `http://www.nordugrid.org/documents/userguide.pdf`.

The Unix command prompt is represented with the dollar sign and text which should be entered by the user is written in typewriter font as follows:

```
$ command
```

Please don't feel restricted by the order of how examples and exercises are presented — explore, edit the xRSL files, try out different commands and parameters and ask questions.

2 Getting started

Basically, a user needs an Internet connection, a web browser, the NorduGrid/ARC User Guide, the NorduGrid User Interface (UI) client software, and grid identity, i.e. user certificate. In tutorials client software and temporary tutorial user certificates are usually pre-installed. If the client software is installed from the standalone package the UI is initialized, i.e. the UI commands are added into user's `$PATH`, etc., by sourcing the setup script:

```
$ cd nordugrid-standalone-0.4.5
$ source setup.sh
```

If you are going through this exercise by yourself, you can find the instructions how to install client software and obtain certificate from NorduGrid website and chapters 3 and 4 of the user guide. *Be prepared that obtaining the personal certificate may take a week.*

The tutorial examples are available as a tar package. Download and uncompress them with:

```
$ wget http://staff.csc.fi/ajt/presentations/
    CSC_ARC_Tutorial_Examples_2005-09-20.tar.gz
$ tar zxvf CSC_ARC_Tutorial_Examples_2005-09-20.tar.gz
```

The examples are in directories

```
dynamic
hellogrid
povray
static
transfers
```

3   Getting information about the resources in NorduGrid

NorduGrid computing clusters and file servers publish information about their available resources using LDAP servers. Contact information to these LDAP servers is collected into the Nordugrid Information System. Although LDAP servers can be explored using the `ldapsearch` command line tool, it is much more convenient to access the information through NorduGrid Grid Monitor's web interface `http://www.nordugrid.org/`. Click on the "Grid Monitor" link at the top of the page. NorduGrid Information System does not require authentication, and all information in it is public, so go ahead and see what's going on!

The main view of the monitor shows currently connected computing resources. Most of the elements are links, clicking on them opens a new window giving more information of that particular resource. For example, click on a cluster name to view more information about that cluster, on the process bar to view more information about jobs running on the cluster, etc. The main view also has small icons to open windows to userbase, storage resources and general search tool. More detailed description of the grid monitor is in the chapter 10 of the User Guide.

Exercises:

- What is the processor type in the Monolith cluster in Sweden? How much memory is installed in the nodes?

- Which version of NorduGrid software and which runtime environments are installed in the Aalborg cluster in Denmark?

- On which clusters is user "Aleksandr Konstantinov" authorized to run jobs?

- Which Storage Elements have more than a terabyte of free disk space?

- Check information about your tutorial identity. Where are you authorized to submit jobs?

4   Logging in into the grid

User certificate is usually stored in directory `$HOME/.globus` in file `usercert.pem` and and the corresponding private key in file `userkey.pem`. You can view information about the certificate with command `grid-cert-info`.

"Logging in" with

```
$ grid-proxy-init
```

creates a temporary access token called proxy, which is discussed in detail in the section 4.2.1 of the user guide. Grid services can act on the behalf of the user only as long as the proxy is valid. Actually, anybody holding the proxy can act as the user.

Exercises:

- Print the certificate in text form by typing `grid-cert-info`. What is your identity in the grid? Who has signed the certificate?

- Logging in to the grid actually means creating a temporary access token called grid proxy. See what the proxy file looks like. Print information of your proxy in clear text by typing `grid-proxy-info`. How long is it valid?

- Some tasks take longer time that the proxy is valid by default. How do you extend the validity time of the proxy?

5 Submitting a simple job

Take a look at file `hellogrid.sh`. It is a simple shell script which writes "Hello Grid" on the standard output and sleeps for a while before returning. You can try to run it locally by typing

```
$ ./hellogrid.sh
```

The job description file to submit this script to the grid is `hellogrid.xrsl`:

```
& (executable=hellogrid.sh)
(stdout=hello.out)
(stderr=hello.err)
(gmlog=gridlog)
(cputime=10)
(memory=32)
(disk=1)
```

Try to submit the job to NorduGrid:

```
$ ngsub -d 1 -f hellogrid.xrsl
```

Command `ngsub` takes a while to complete as UI first contacts the root information server, asks for clusters connected at the moment and then queries all the available clusters individually for their attributes etc. Optional flag `-d 1` in `ngsub` command let's you view the progress of this procedure.

UI selects suitable host for the job based on the information that it gathered from the clusters. Then it parses the xRSL job description and sends it to the selected cluster possibly along with the input files that are stored locally in the UI client machine. When the job is submitted, you should receive a message such as

```
Job submitted with jobid
  gsiftp://benedict.aau.dk:2811/jobs/2837896291031006429
```

In this case, the job was submitted to `benedict.aau.dk` in Denmark and the url `gsiftp://benedict.aau.dk:2811/jobs/2837896291031006429` is the reference to the job. The last part is a session directory chosen randomly by the target system. It is possible to check the status of the job using the `ngstat` command:

```
$ ngstat gsiftp://benedict.aau.dk:2811/jobs/2837896291031006429
Job gsiftp://benedict.aau.dk:2811/jobs/2837896291031006429
  Jobname: hellogrid
  Status: FINISHED 2004-03-29 16:15:18
```

In this case the job has been successfully completed. Other stages that the job may be in are described in the NorduGrid User Guide. Retrieve the results by typing

```
$ ngget gsiftp://benedict.aau.dk:2811/jobs/2837896291031006429
```

This downloads the result files and some statistics in the directory 2837896291031006429. Take a look at the output (files `stdout` and `stderr` and the files in the `gridlog` directory. What can you see?

Notice, that we made no reference to which cluster the job should go. If you would like to specify the cluster (or exclude some), it can be described in the xRSL file or on the command line:

```
$ ngsub -f hellogrid.xrsl -c kivi.csc.fi
```

Exercises:

- Specify three alternative clusters as accepted targets in the CH4_LUCKY.xrsl file. Try submitting the job. (Hint: Use the "`cluster`" attribute, see the User Guide for details.)

- Can you store a list of preferred/discarded clusters in a separate file and give that list to `ngsub`?

- Submitting the job with command `ngsub -d 1 -f hellogrid.xrsl` shows information about the submission process. What more info is available with `-d 2`?

- Specify a job name by adding line (`jobname=hellogrid_your_name`) to the file `hellogrid.xrsl`. Submit the job again. Now you can refer to the job with the name instead of session name url when using `ngstat` and `ngget` commands.

- Submit some more jobs and try commands `ngkill` and `ngclean`.

6   Simple file transfers

The set of exercises in this section demonstrates the use of simple file transfer tools in NorduGrid ARC middleware. Here "simple" means point-to-point file transfers in which the file locations are explicitly specified by the user. Simple file transfers can be made using interactive tools, such as gsincftp. File transfers can also be initiated from the client side by `ngsub` or from the server side by grid-manager as described in the job description. This section does not cover more advanced data management and indexing services, such as Globus RLS, Globus Replica Catalog or EGEE/gLite Fireman. Data management is discussed in User Guide's chapter 9.

In the simplest case all the input files for a job are submitted from the local machine running user client software along with the job description xRSL file, and the results are downloaded from the computing resource's session directory to the local client machine by the user. The examples in the previous section use this model.
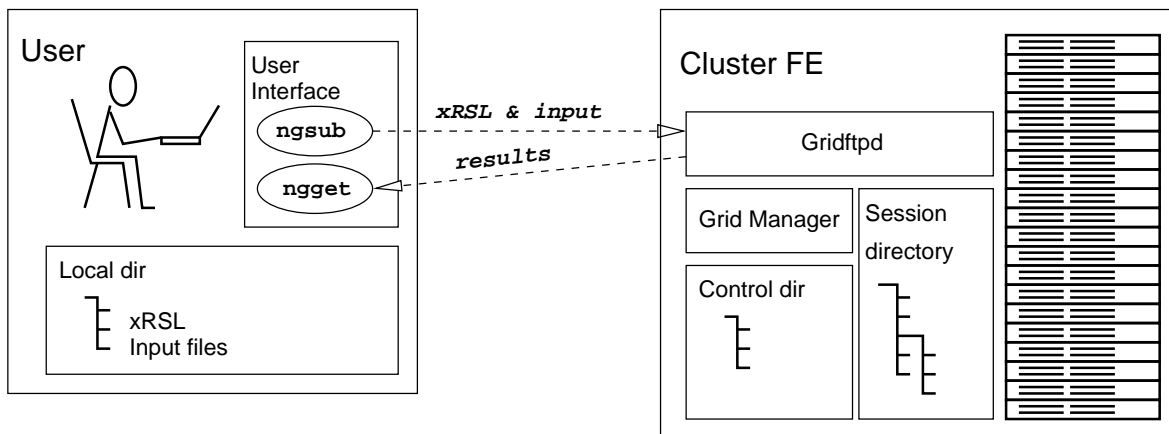
Figure 1: Simple file transfers initiated by UI and user.

The session directories are kept on the computing resources for a limited time only, usually at least for 24 hours. Client machines could be laptops etc., and are not necessarily connected to the grid when the jobs finish, so the grid-manager does not transfer jobs directly back to the client machine. Persistent storage areas (file servers) which are continuously connected to grid and can accept the output of the jobs automatically are called storage elements (SE).

Storage elements can be accessed interactively. Let's first find a SE, using Grid Monitor, in which our tutorial identity is authorized. Add

```
(outputfiles =
  ("hello.out" "gsiftp://se1.ndgf.csc.fi/ndgf/tutorial/<dirname>/hello.out"))
```

to the `hellogrid.xrsl` file from the previous section. Open a connection to the SE and create a directory <dirname> for yourself:

```
$ gsincftp gsiftp://se1.ndgf.csc.fi/ndgf/tutorial
$ mkdir <dirname>
$ quit
```

Now, when you submit the `hellogrid.xrsl` example the output file `stdout.txt` is automatically moved to the specified destination by the grid-manager after the job has finished. If proxy has expired before the job finishes, the file transfers from the computing element to SE fail and the output files stay on the computing element.

File permissions on SEs can be controlled by several schemes. Basically the question is about how to map grid identities and permissions to local unix login accounts and file permissions. In this example we discuss the Grid Access Control List (GACL) scheme. In the GACL scheme each file and directory is accompanied by the GACL file describing the grid access permissions. You can view the content of the GACL file with

```
$ ngacl get gsiftp://se1.ndgf.csc.fi/ndgf/tutorial/<dirname>/<filename>
```
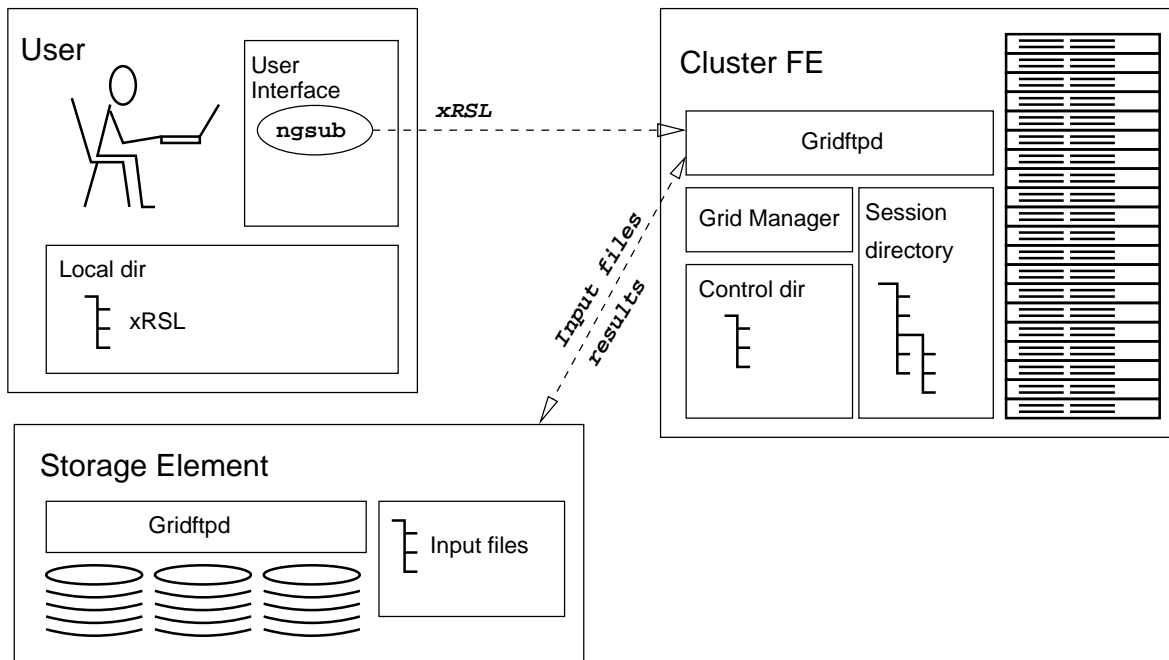
Figure 2: Simple file transfers initiated by UI and grid-manager.

Changing the permissions involves changing the GACL file. This can be done using ARC UI tool `ngacl`.

There are a number of options controlling the file transfer itself, such as file caching and file encryption. Please refer to the section 7.1.1 in the User Guide.

Exercises:

- Also executables and other input files can be downloaded to the computing resource from storage elements prior to the execution of the job. Move `hellogrid.sh` to a SE and add corresponding `inputfiles` attribute to the `hellogrid.xrsl` file. Submit the job.

- Nothing in principle forbids one to install NorduGrid file server on the client machine, to make it act as a private SE, which stores files with local user account permissions directly. It would make a nice exercise, but it probably takes a bit more time than is available in usual tutorial sessions.

- Try ARC UI command line tools `ngls`, `ngremove (ngrm)`, `ngcopy (ngcp)` in place of gsincftp.

- How do you find out what access control scheme a SE uses? Hint: Use the Grid Monitor Search tool.

- Usually the SEs are configured so that only the creator of the file has any permissions on it. Give a read access to the directory you created and to one of your files in the SE to someone else in the tutorial session.

- Where did you find documentation about GACL? ;-)

7 Statically linked executable

This example demonstrates how to run a simple serial computation on the grid. The application is a first-principles real-space electronic structure program calculating the electronic structure of the CH4 molecule. Thanks to Tuomas Torsti for providing the example. In this case the (statically linked) executable is submitted to the grid as one of the job input files and no reference to Runtime Environments (software packages installed on the target cluster) is required. Basically we request a single i386 compatible PC.

Go to directory containing the material:

```
$ cd static
$ ls
CH4_LUCKY.xrsl  INPUT  potentials  rspace-0.81_i386-linux_SERIAL
```

The job description is in the file CH4_LUCKY.xrsl:

```
$ cat CH4_LUCKY.xrsl
&(executable=rspace-0.81_i386-linux_SERIAL)
(JobName=CH4_LUCKY)
(inputFiles=(INPUT "")
            (potentials/C "")
            (potentials/H ""))
(outputFiles=(energies "")
             (forces "")
             (WAVES_1 "")
             (POTENTIAL ""))
(CpuTime=10)
(memory=64)
(disk=10)
(stdout=stdout.txt)
(stderr=stderr.txt)
(gmlog=debugdir)
(|(architecture=i386)
  (architecture=i686)
  (architecture=x86_64))
```

First line defines the name of the executable. If it is not specified in the list of input files, it is automatically appended there. Edit the job name from CH4_LUCKY to CH4_LUCKY_YOUR_FIRST_NAME so you can differentiate the instance submitted by you from the others in the tutorial more easily.

Read from the User Guide how the location of the input and output files is resolved. That can be tricky with all the available locations...

Next some of the requirements for the job are specified, so that the user interface can select a suitable platform (cluster).

Submit the job!

```
$ ngsub -f CH4_LUCKY.xrsl
INPUT->INPUT     1 s:           0 kB          0 kB/s          0 kB/s     . . .
rspace-0.81_i386-linux_SERIAL->rspace-0.81_i386-linux_SERIAL    1 s:     . . .
rspace-0.81_i386-linux_SERIAL->rspace-0.81_i386-linux_SERIAL    2 s:     . . .
C->C    1 s:           0 kB          0 kB/s          0 kB/s    . . .
C->C    2 s:          64 kB         31 kB/s         32 kB/s    . . .
H->H    1 s:           0 kB          0 kB/s          0 kB/s    . . .
Job submitted with jobid gsiftp://ingvar.nsc.liu.se:2811/jobs/7009965451436415513
```

Monitor the job with `ngstat` and when it is finished, fetch the results with `ngget`.

Exercises:

- Add a "`notify`" attribute in the xRSL file to receive email notifications of job status changes. See the User Guide for details.

- If one does not retrieve or clean finished jobs from computing resource, resourse cleans them by itself after some time. However, UI "remembers" these jobs and when running `ngstat -a`, it complains about jobs that are not found. UI's list of sent jobs can be refreshed from the Information System with `ngsync`, which is useful also if one is moving from a machine to another. Where does UI save the IDs of the submitted jobs?

8 Dynamically linked executable

In the previous example the executable was a statically linked binary. Running such binaries requires that the job goes to machine with suitable architecture. If the executable is dynamically linked, a successful execution of the job also requires that all the necessary libraries are available. This is can be achieved in at least two ways. The first is to submit libraries and the linker (!) as input files – a case which is quite close to submitting statically linked binary. The second option is to install libraries on the computing resource and advertise them in the Information System using Runtime Environments. This section demonstrates the use of the first approach.

Below is a wrapper script for a laminate structure optimization run

```
#!/bin/sh -v
#$ -cwd

export ELMER_HOME=.

tar -p -z -x -f unchangebles.tar.gz
./lib/ld-linux.so.3 --library-path "./lib" ./ElmerSolver
```

and the corresponding job description:

```
&(rsl_substitution =  (SEDIR
             "gsiftp://se1.ndgf.csc.fi/ndgf/tutorial/dynamic" ))
```

```
(jobName      = "elmer" )
(executable   = "wrapper.sh" )
(inputfiles   = (wrapper.sh "" )
                (laminate.opt "" )
                (Shell.sif "" )
                (unchangebles.tar.gz $(SEDIR)/unchangebles.tar.gz ) )
(executables = lib/ld-linux.so.3
                lib/elements.def
                ElmerSolver lib/ld-linux.so.3
                lib/libc.so.6
                lib/libcxa.so.3
                lib/libg2c.so.0
                lib/libSolver.so
                Shell)
(outputfiles = (layup.opt "" )
                ("/" "" ) )
(stdout      = stdout.txt )
(stderr      = stderr.txt )
(gmlog       = logs )
(cache       = yes )
(disk        = 150 )
(cpuTime     = 3 )
(|(architecture = i686 )
  (architecture = i386 )
  (architecture = x86_64))
```

This job description directs grid-manager to download a largish tar package unchangebles.tar.gz
containing linux loader, executable, and the required libraries. Shell wrapper untars the package and
runs the executable with specified library path.

Exercises:

- One can monitor the progress of a job while it is running with ngcat, which prints the standard
  output of the job. Try it. Quite useful.

9 Using a Runtime Environment

Runtime Environments provide a means to make software packages installed at the systems available
on the Grid. Users can specify in the job description file that a specific runtime environment needs to
be present in the target system. That avoids the need to send the actual application binary as part of
the computation: only input files need to be sent.

The following tiny script and job description file can be used to render an image using the Persistence
of Vision Raytracer (POV-Ray) tool.

File runpov.sh:

```
#! /bin/sh
povray $@
```

File `povrayjob.xrsl`:

```
&(executable=runpov.sh)
(arguments="-d -W640 -H480 skyvase.pov")
(stdout="pov.out")
(stderr="pov.err")
(gmlog="log")
(jobname="povray-skyvase")
(runtimeenvironment=ADD_A_SUITABLE_RE_HERE)
(cputime=10)
(inputfiles=(skyvase.pov ""))
(outputfiles=(skyvase.png ""))
```

Exercises:

- Take a look at the Runtime Environment Registry at `http://www.csc.fi/grid/rer/` and find the most recent runtime environment to run POV-Ray jobs. Modify the job description file accordingly and submit the job. You can also try different parameters for rendering.

- Which clusters have some version of the POVRAY runtime environment installed? Extend the job description file so that also older versions of POV-Ray are accepted.

- Try rendering some of the other images and scenes in the povray subdirectory. Some of them require more than one input file and are placed in separate directories.

10   Creating complex jobs

Real jobs normally send several subjobs to be executed on the Grid. One job shouldn't be too long to avoid losing a lot of time in case of job failure (besides, most sites have a maximum time limit in the queue) but it shouldn't be too short either to reduce the overhead resulting from submitting the job and retrieving the results.

Depending on the type of problem, the user sometimes needs to split a large computation into smaller parts and later combine the results, and sometimes group small computations into larger chunks. This exercise presents both approaches.

The following script uses POV-Ray to render three images as one job.

```
#!/bin/sh
povray -d -ochair.png chair.pov
povray -d -odrink.png drink.pov
povray -d -oegg.png egg.pov
```

Exercises:

- Write a job description file to run the job. Submit it and retrieve the results.

- Look at the execution time in the log files. Was the lenght of the job suitable to be efficiently executed in the Grid?

The following job description file and wrapper script uses POV-Ray to render a five frame animation. The parameter `-kff` specifies the number of frames, parameter `-ki` specifies clock value for the initial frame of the animation and parameter `-kf` specifies the clock value for the final frame of the animation.

File `fractalzoom_part1.sh`:

```
#!/bin/sh
povray -d -kff5 -ki0.00 -kf0.08 -ofractalzoom_part1_frame fractalzoom.pov
tar czvf fractalzoom_part1_frames.tar.gz fractalzoom_part1_frame*.png
```

File `fractalzoom_part1.xrsl`:

```
&(executable=fractalzoom_part1.sh)
(stdout="fractalzoom.out")
(stderr="fractalzoom.err")
(gmlog="log")
(jobname="povray-fractalzoom")
(|(runtimeenvironment=APPS/GRAPH/POVRAY-3.6)
  (runtimeenvironment=POVRAY-3.5))
(cputime=10)
(inputfiles=(fractalzoom_part1.sh "")
            (fractalzoom.pov ""))
(outputfiles=(fractalzoom_part1_frames.tar.gz ""))
```

Exercises:

- Try to submit the job and see what kind of result files are produced. Among other files there should be a tar package containing five images. Watch the result as an animation by extracting all the images in the same directory running the command `animate -delay 50 *.png`.

- Write additional jobs and description files for rendering subsequents parts of the animation for time periods 0.10 - 0.18, 0.20 - 0.28, 0.30 - 0.38 and 0.40 - 0.48. Change the filename format specified by `-o` option in the wrapper script slightly to produce different filenames in each run.

- Multiple jobs can be described in single job description file and submitted with one ngsub command. This saves some time because then ngsub goes through the list of available hosts only once. Combine the five xRSL files of the previous exercise into one xRSL file and submit the job again.

To see a bit more complicated wrapper script to split a POV-Ray scene to several parts, submit them in the grid and automatically fetch and combine the results, look at Leif Nixon's example at `http://www.nsc.liu.se/~nixon/ng-povray/`.

11   Submitting your own application

You can also try your own application to NorduGrid. It is easiest if you have a statically linked executable (compiled for Linux x86 platform) which does not require any software to be installed beforehand in the target cluster. Write a job description file and submit the job, monitor the progress and fetch the results.

12   Acknowledgements

The POV-Ray scenes in the examples package include images created by Gilles Tran, available under the Creative Commons Attribution License (`http://creativecommons.org/licenses/by/2.0/`).