

ARC Middleware Tutorial Exercises

Notur2008, Tromsø, Norway
June 3rd, 2008

Ivan Degtyarenko and Olli Tourunen
CSC – the Finnish IT Center for Science &
Nordic Data Grid Facility

Table of Contents

1. Introduction.....	3
2. Getting started.....	3
3. Finding information about resources available on NorduGrid.....	4
4. Logging in to the grid.....	4
5. Submitting a simple job.....	5
6. Simple file transfers.....	7
7. Statically linked executable.....	10
8. Dynamically linked executable.....	11
9. Using Runtime Environments.....	12
10. Using the BLAST Runtime Environment.....	14
11. MPI Runtime Environment.....	14
12. Acknowledgments.....	15

1. Introduction

This document contains examples and exercises for the NorduGrid ARC middleware tutorial. A copy of the NorduGrid ARC User Guide should be used side by 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 in which the examples and exercises are presented. Feel free to explore, edit the xRSL files, try out different commands and parameters. Do not hesitate to 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 a grid identity (i.e. user certificate). In tutorials the client software and temporary tutorial user certificates are usually pre-installed. If the client software is installed from the standalone package, the UI will need to be initialized by sourcing the setup script. This will add the UI commands to the user's \$PATH, set environment variables, etc.:

```
$ cd nordugrid-arc-standalone-0.6.1-i386
$ source setup.sh
```

Note: the above is not needed for Notur2008 tutorial, as the client is installed system wide.

If you are going through these exercises by yourself, you can find the instructions how to install the client software and obtain a certificate from the NorduGrid website from chapters 3 and 4 of the User Guide. Be prepared that obtaining the personal certificate may take up to two weeks, during the holiday season possibly longer.

The tutorial examples and material will be made available on the course archive web page. The examples have been preinstalled in the home directories of your workstation.

The examples are in the following directories

```
$ ls examples
blast dynamic hellogrid openmpi povray static transfers
```

3. Finding information about resources available on 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 the Grid Monitor's web interface <http://www.nordugrid.org/>. Click on the "Grid Monitor" link at the top of the page. The NorduGrid Information System does not require authentication. All the 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 about that particular resource. For example, try the following:

- click on a cluster name to view more information about that cluster.
- click on the process bar to view more information about jobs running on the cluster

The main view also has small icons to open windows to userbase, storage resources and a general search tool. A more detailed description of the grid monitor is in chapter 10 of the User Guide.

Exercises:

1. What is the processor type in the Stallo cluster in Tromsø? How much memory is installed in the nodes?
2. What are the top 5 clusters according to the number of processors?
3. Which version of NorduGrid software and which runtime environments are installed in the Akaatti cluster in Finland?
4. On which clusters is the user "Olli Tourunen" authorized to run jobs?
5. Which Storage Elements have more than a terabyte of free disk space?

4. Logging in to the grid

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

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

This creates a temporary access token called a 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:

1. Print the certificate in text form by typing `grid-cert-info`.
 - What is your identity in the grid?
 - Who has signed the certificate (Issuer field)?
2. Logging in to the grid actually means creating a temporary access token called a grid proxy.
 - Print information of your proxy in clear text by typing `grid-proxy-info`.
 - How long is it valid?
 - See also what the proxy file looks like. (for example with `cat`)

3. Some tasks take longer than the default validity period of the proxy.

- How can 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 running it locally by typing the following:

```
$ cd examples/hellogrid
$ ./hellogrid.sh
```

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

```
& (executable=hellogrid.sh)
(stdout=hello.out)
(stderr=hello.err)
(gmlog=gridlog)
(cputime=10)
(memory=200)
(disk=1)
```

Try to submit the job to NorduGrid with the following command:

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

The command `ngsb` takes a while to complete as the 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. The optional flag `-d 1` in the `ngsb` command lets you view the progress of this procedure. The UI parses the xRSL job description and then selects a suitable host for job execution. The decision is based on the information that was gathered from the clusters and requirements stated in the job description file. Then the UI sends the job 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 of the URL 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
```

Note: the job is identified by the whole URL, not just the numeric part. You can also use `ngstat -a` to view the status of all your jobs. In the case above the job has been successfully completed. Job states are described in detail in chapter 4 of the *The NorduGrid Grid Manager And GridFTP Server: Description And Administrator's Manual*"

You can 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 `hello.out` and `hello.err`) and the files in the `gridlog` directory. What do 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 specified in the `xRSL` file or on the command line:

```
$ ngsbub -f hellogrid.xrsl -c kiniini.csc.fi
```

```
$ ngsbub -f hellogrid.xrsl -c -kiniini.csc.fi -c -jaspis.hip.fi
```

Note: Set the memory requirements (`memory=200`) high enough. Some batch queue systems include their own processes in the job memory consumption.

Exercises:

1. Specify a job name by adding the 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.
2. Specify two alternative clusters as accepted targets in the `hellogrid.xrsl` file. Try submitting the job. (Hint: Use the "cluster" attribute, see the User Guide for details.)
3. Submitting the job with command `ngsbub -d 1 -f hellogrid.xrsl` shows information about the submission process. What more info is available with `-d 2`?
4. Submit a few more jobs and try the commands `ngkill` and `ngclean`.

6. Simple file transfers

This set of exercises demonstrates the use of simple file transfer tools in the 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 by using interactive tools, such as `uberftp`. File transfers can also be initiated from the client side by `ngsbub` or from the server side by the grid-manager as described in the `xRSL` file. This section does not cover more advanced data management and indexing services, such as

Globus RLS, Globus Replica Catalog or EGEE/gLite Fireman. However, this straightforward approach can be quite powerful when used in an intelligent way. Data management is discussed in chapter 9 of the User Guide.

In the simplest case all the input files for a job are sent by the user from the local machine along with the job's xRSL file, and the results are downloaded from the computing resource's session directory to the local client machine by the user.

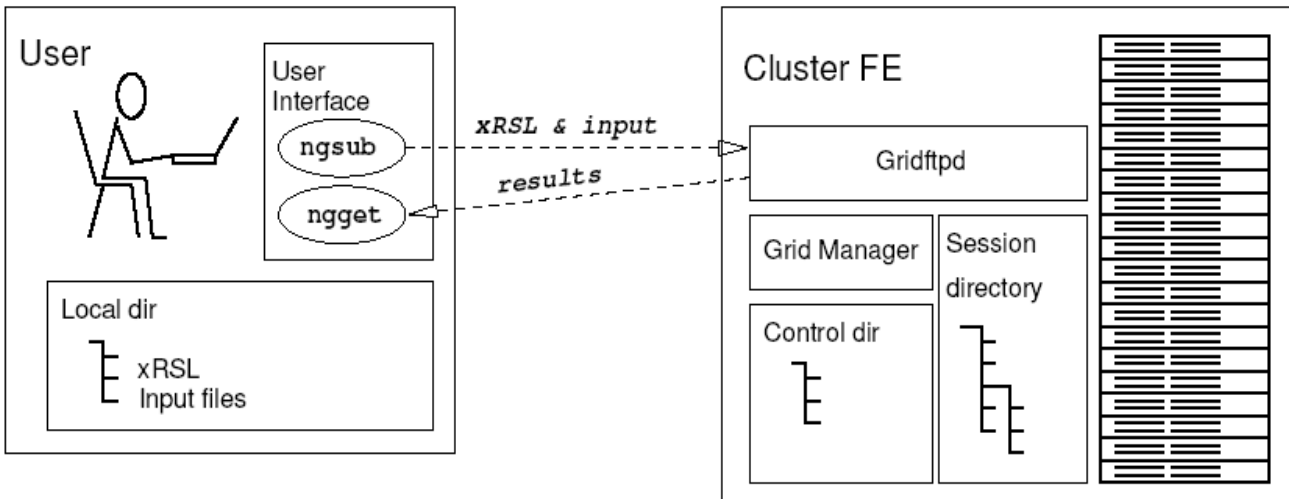


Figure 1: Simple file transfers initiated by the UI and the user

The examples in the previous section use this model. The session directories are kept on the computing resources for a limited time only, usually for at least for 24 hours, typically 2 weeks. Client machines can be laptops etc., and are not necessarily connected to the grid when the jobs finish. Because of this 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 are called storage elements (SE). Storage elements can also be accessed interactively. Let's first find a SE in which our tutorial identity is authorized using the Grid Monitor (hint: use the search tool and give your grid identity as a parameter). We will most probably only be allowed to access the se1.ndgf.csc.fi storage element. We want to store our job output in the SE. Add the following:

```
(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 (<dirname> being e.g. your tutorial identity userNN):

```
$ uberftp se1.ndgf.csc.fi
$ cd /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 the proxy has expired before the job finishes, the file transfers from the computing element to the SE fail and the output files stay on the computing element.

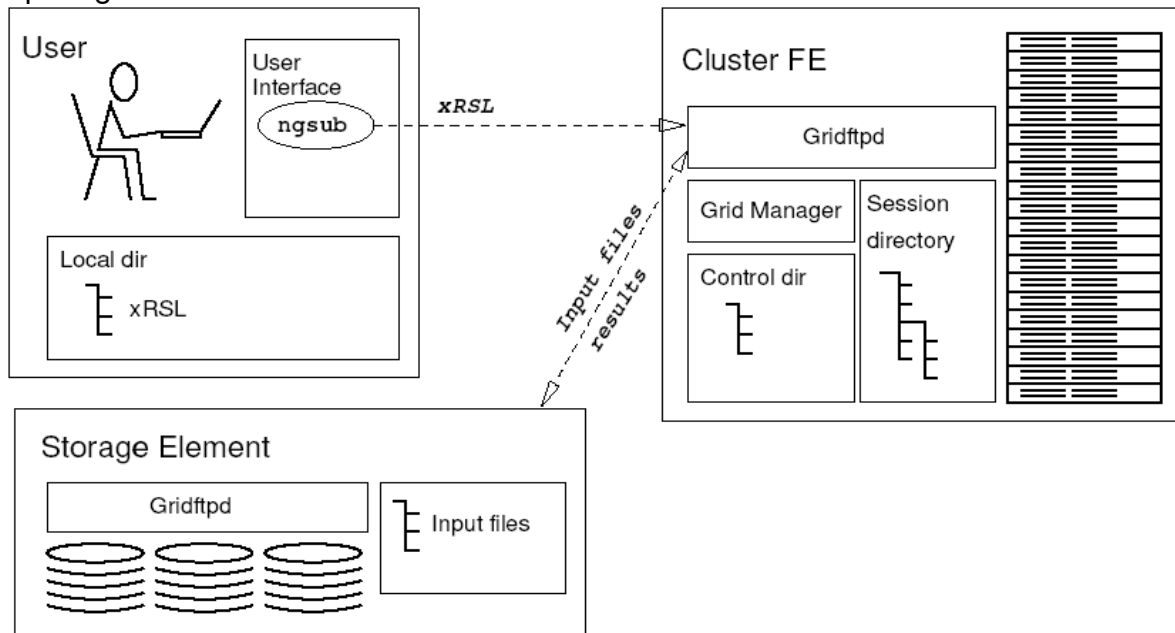


Figure 2: Simple file transfers initiated by the UI and the grid-manager

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.

File permissions on the SEs can be controlled by several schemes. Basically the question is about how to map the grid identities and permissions to the local unix login accounts and file permissions. In this example we use the Grid Access Control List (GACL) scheme. In the GACL scheme each file and directory is accompanied by a GACL file describing the grid access permissions. Changing the permissions involves changing the GACL file. This can be done using ARC UI tool `ngacl`. You can view the content of the GACL file with

```
$ ngacl get gsiftp://se1.ndgf.csc.fi/ndgf/tutorial/<dirname>/<filename>
```

Exercises:

1. Executables and other input files can also be downloaded from storage elements by a computing resource prior to the execution of a job (see Figure 2). Upload `hellogrid.sh` to a SE and add the corresponding `inputfiles` attribute to the `hellogrid.xrsl` file. Submit the job.
2. Try ARC UI command line tools `ngls`, `ngremove` (`ngrm`), `ngcopy` (`ngcp`) in place of `uberftp`.
3. How do you find out what access control scheme a SE uses? Hint: Use the Grid Monitor Search tool.
4. Advanced: Usually the SEs are configured so that only the creator of the file has permissions to access it. Try giving read access to the directory

you created and to one of your files on the SE to someone else in the tutorial.

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 (methane) molecule. Thanks to Tuomas Torsti for providing the example. In this case the (statically linked) executable is simply submitted to the grid as one of the job input files. Basically we request a single i386 compatible PC.

Change to the static directory:

```
$ 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=200)
(disk=10)
(stdout=stdout.txt)
(stderr=stderr.txt)
(gmlog=debugdir)
(|(architecture=i386)
 (architecture=i686)
 (architecture=x86_64))
```

The 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 distinguish the job you have submitted from those submitted by others. For documentation of the inputFiles and outputFiles attributes, see the User Guide.

At the end of the job description the requirements for the computing element are specified, so that the user interface can select a suitable platform (cluster). Submit the job! (-d 1 used in this example to show a bit more about what's going on)

```
$ ngsb -d 1 -f CH4_LUCKY.xrsl
Proxy subject name: /C=FI/L=Espoo/O=CSC/CN=Notur2008 Tutorial15/CN=1691871522
Proxy valid to: 2008-06-03 23:29:24
```

```

Proxy valid for: 12 hours, 45 minutes, 9 seconds
Queue selected: mgrid@kivi.csc.fi
File uploaded: /tmp/tourunen/rsl.uze6h1
File uploaded: /home/tourunen/examples/static/INPUT
File uploaded: /home/tourunen/examples/static/potentials/C
File uploaded: /home/tourunen/examples/static/potentials/H
File uploaded: /home/tourunen/examples/static/rspace-0.81_i386-linux_SERIAL
Job submitted with jobid:
gsiftp://kivi.csc.fi:2811/jobs/2668311807054591238131027
$

```

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

1. If one does not retrieve or clean finished jobs from a computing resource, the resource cleans them by itself after some time (a week or two by default). However, the UI "remembers" these jobs and when running `ngstat -a`, it will complain about jobs that no longer exist. The UI's list of sent jobs can be refreshed from the Information System with the command `ngsync`. `ngsync` is also useful if one is moving from one 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 a machine with a suitable architecture. If the executable is dynamically linked, successful execution of the job requires that all the necessary libraries are available. This is can be achieved in at least two ways. The first is to submit the libraries and the linker (!) as input files - a case which is quite close to submitting a statically linked binary. The second option is to install the libraries on the computing resource and advertise them on the Information System using Runtime Environments. This section demonstrates the use of the first approach. The examples are located in the `dynamic` subdirectory.

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://sel.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 the Linux loader, an executable and the required libraries. The shell wrapper `untars` the package and runs the executable with the specified library path.

Exercises:

1. One can monitor the progress of a job while it is running with the command `ngcat`. This prints the standard output of the job. It is a quite useful feature, and not possible with all grid middleware. Use `ngcat -e` to print the standard error output.

9. Using Runtime Environments

Runtime Environments (REs) provide a means to advertise the availability of software packages installed on the computing resources on the grid. In the job description users can require that a specific RE is present on the target system. That avoids the need to send the actual application binary as part of the job: only the input files need to be sent.

The following tiny script (located in the `povray` subdirectory) 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:

1. Take a look at the Runtime Environment Registry at <http://gridrer.csc.fi> 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.
2. Which clusters have some version of the POV-Ray runtime environment installed? Extend the job description file so that also older versions of POV-Ray are accepted.
3. 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 subdirectories.
4. Advanced: Try using the Elmer Runtime Environment for the job in the previous chapter instead of dynamic linking.

10. BLAST Runtime Environment

NCBI BLAST is common software used in the biosciences to find similarities or matches in genetic sequences. This example demonstrates that runtime environments can also provide additional services such as preparing input files in an optimal way for the local cluster and hide parallel execution completely from the user.

The xrsl file specifies that BLAST software should be present on the cluster on which it will run. Also, the database that the query is run against is transferred from a storage element and can be shared between different users. In this example BLAST compares 10 sequences to the Uniprot-Swissprot-database and returns a scored list of matches. Notice also how we use xrsl to hide the differences between the input files thus eliminating the need to change the job script.

1. Try to find out whether the shared database is really transferred from the storage element or does it come from the server's cache. Hints:
 - a) at least kiiini.csc.fi supports caching
 - b) a well placed `ls -l` in the job script might tell you something along with the server logs in the gmlog -subdirectory in the job's results.
2. Modify the job description to disable caching.

11. MPI Runtime Environment

An OpenMPI runtime environment has been installed for test use on some of the M-grid clusters. In the `openmpi` directory you will find a binary program for calculating PI, a submission script and a job description file. The program has been compiled with 64bit GCC and the OpenMPI libraries and it takes 1 argument "Epsilon" which is a "small number."

To run your program in the OpenMPI runtime environment, you must execute a short shell script which runs the program.

File `pi_openmpi_gnu64.sh`:

```
/usr/bin/time -p $MPIRUN $MPIARGS pi_openmpi_gnu64.bin $*
```

In the job description file the shell script (`pi_openmpi_gnu64.sh`) is specified as the called executable. The binary program (`pi_openmpi_gnu64.bin`) is an inputfile and should also be listed under executables so that the correct permissions are set when it is copied to the cluster. This program does not generate any output files other than stdout and stderr (note: the output of `/usr/bin/time` goes to stderr) and gmlog. The count parameter is used to specify the number of processors to use. Since the program needs at least three processes, count should be at least 3.

File `pi_openmpi_gnu64.xrsl`:

```
&(runtimeenvironment="ENV/MPI/OPENMPI-1.2/GCC64-TEST")  
(jobName="PI calculation with OpenMPI RE")
```

```
(count=3)
(cpuTime="5 minutes")
(executable="pi_openmpi_gnu64.sh")
(arguments="1E-6")
(inputfiles=("pi_openmpi_gnu64.bin" ""))
(executables="pi_openmpi_gnu64.bin")
(stdout="stdout")
(stderr="stderr")
(gmlog="gmlog")
```

Exercises:

1. Find the MPI runtime environment from the Runtime Environment Registry.
2. Submit the program with count set to 3
3. Try submitting the job with different values for count and Epsilon.
4. Try viewing the results in stdout before the job finishes with "ngcat jobid"
5. Try viewing stderr before the job finishes with "ngcat -e jobid"

12. Acknowledgments

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/>).

This material is based on the original work of Arto Teräs and Juha Lento. Thank you! Updates and new material added by Michael Gindonis and Olli Tourunen.

Comments and corrections can be sent to olli.tourunen at csc dot fi.