Introduction

The Grid is a *distributed architecture* for delivering computing and data resources over the internet. The word Grid is chosen by analogy with the *electric power grid*. According to this analogy, the end user does not have to know or care where the computing is performed or the data stored. The software tools (*middleware*) developed for the Grid are suitable for remote instrumentation and virtual environments. Rapid development of new technologies enabled the forming of the Grid: "Scientific computing is getting more and more data intensive. Simultaneously, the demand for applications that are easy to use grows as new branches of science are taking computing methods into use. The *development of networks* has made it possible to offer versatile information services regardless of their physical locations. In the intersection of these trends a new concept has been created: the Grid."

The Grid is applicable to tasks where *widely distributed resources* are available on a Wide Area Network (WAN). The Grid problem can be defined as flexible, secure and manageable sharing of the computing resources in a virtual organization (VO). This kind of an organization consists of a dynamic group of individuals, institutions and resources. Grid projects are *multi-organizational*.

A *metacomputer* is a collection of computers in different physical locations that can be used like a local computer. In essence, the Grid concept is a logical continuation of the metacomputer. The main difference between the metacomputer and Grid concepts is that, in a metacomputer, the resources are usually connected using a Local Area Network. For a Grid, the resources are distributed on a Wide Area Network. As a consequence, the data processing and saving for a Grid (or at least their manageability) is genuinely distributed on a national, international, and even global level.

Grid computing

The Grid empire is expanding fast, particularly across all borders - a lot of new Grid research programs and initiatives are being established all over the world and many of those are growing rapidly. The overall goal of all these programs is to make Grid computing as universally accepted and standardized as electricity is now.

In HEP one of the most challenging tasks is to build and maintain a data storage and analysis infrastructure for the LHC. The LHC will produce roughly *15 Petabytes* (1.5×10^7GBytes) of data annually. Thousands of scientists around the world will access and analyse those data. In addition, all the data needs to be available over the estimated *15-year lifetime of LHC*. The analysis of the data, including comparison with theoretical simulations, requires of the order of 100,000 CPUs at 2004 measures of processing power. A novel globally distributed model for data storage and analysis was chosen - a computing Grid.

Grid middleware

The Grid relies on advanced software called middleware, which ensures *seamless communication* between different computers and different parts of the world. Grid middleware refers to the *security, resource management, data access, instrumentation, policy, accounting, and other services* required for applications, users, and resource providers to operate effectively in a Grid environment. Middleware acts as a sort of 'glue' which binds these services together. To deploy Grid middleware as a user, a valid *Grid user certificate* is needed.

The middleware can be in general categorized into site services and Virtual Organization (VO) services. The site services consist of security, computing element providing Grid interfaces, storage element providing Grid interfaces to site storage, monitoring and accounting services for inspecting the status of Grid services, VO membership service, workload management providing facilities to

manage jobs, file catalogues for locating and accessing data, information services for publishing and maintaining data, and file transfer services.

During the past few years, numerous Grid and Grid-like middleware products have emerged. Examples include UNICORE, ARC, EDG/LCG-2/gLite, Globus, Condor, VDT and SRB. They are capable of providing some of the fundamental Grid services, such as Grid job submission and management, Grid data management and Grid information services. Unfortunately no widely accepted, implemented and usable standards exist.

The LHC computing Grid project (LCG) develops middleware based on Globus, Condor, Virtual Data Toolkit and gLite; the middleware is EDG/LCG-2/gLite.

The middleware of the NorduGrid is ARC. NorduGrid is the Grid of the Nordic countries.

Usage of ARC

The web page for NorduGrid and ARC middleware is

www.nordugrid.org

where one can find for example the user guide (NORDUGRID-MANUAL-13).

The client software is pre-installed in the kale cluster. First one needs to login in to the Grid, type

```
arcproxy (or voms-proxy-init)
```

and give the pass word. This creates a temporary token called proxy. Grid services can act only as long as the proxy is valid. You can print information of your proxy, like how long it is valid, by typing

```
arcproxy --info (or voms-proxy-info)
```

Submitting a job. Let us create a small job script which prints "Hello World!!", and submit that to the Grid. To do that we need the job script, and a job description file. In the job description file

one has the following lines (JSDL language). File `test.jsdl`:

```
<JobDescription> <JobIdentification>
<JobName>Hello World job</..
</JobIdentification> <Application>
<posix:POSIXApplication>
<posix:Executable>test.csh</..
<posix:Output>test.out</..
<posix:Error>test.err</..
```

To submit the job, type

```
arcsub test.jsdl
```

This will give you a jobid, which you can use for checking the job status

```
arcstat http://...
```

The CE specifies the cluster where the job should run.

```
arcsub test.jsdl -c kale-cms.hip.fi
```

You can kill jobs with command arckill, for cleaning use `arcclean`.

Job monitoring. To monitor the progressing of the job,

```
arccat http://...
```

This command works as cat command. To retrieve the output files,

```
arcget http://...
```

Files to be retrieved by `arcget` should be tagged with DeleteOnTermination=false in the JSDL file.

Cancelling a job is done with `arckill` command. A job can be killed practically on any stage of processing through the Grid.

```
arckill http://...
```

```
arckill -all
```

Jobs can be resubmitted with `arcresub` command. Upon resubmision the job will receive a new job ID. By specifying the –all option, all active jobs appearing the in job list file will be resubmitted.

Grid computing

If a job fails, or you are not willing to retrieve the results for some reason, a good practice for users is not to wait for the Grid manager to clean up the job leftovers, but to use arcclean to release disk space.

```
arcclean http://...
```

If the user proxy expires while the job is still running, new proxy can be uploaded with command

```
arcrenew http://...
```

Example: let us make a program which produces a rootfile, submit the job to the Grid and retrieve the output (the rootfile).

The program used is "writing" in Ex6.1. It writes gaussian distributed random numbers in a tree, which is written in a rootfile.

The problem now is runtime libraries which either need to be present in the Grid worknode, or we must send the libraries with the exe file.
Since we cant be sure that the root version we are using is available in the working node, let us choose the latter option and send the libraries with the executable.

The libraries are copied in a new directory libs, which is then tarred and gzipped. Some files from $(ROOTSYS)/etc may also be needed, depending on the used ROOT version.
The job script needs to unpack the tarball: file `test.job`

```
#!/bin/sh
export ROOTSYS=.
export LD_LIBRARY_PATH=$(ROOTSYS)/libs
tar xfvz libs.tar.gz
./writing
```

For some reason the working node accepted only bash/sh, so let's use that

The environment variables ROOTSYS and LD_LIBRARY_PATH needed to be set, without them the program crashes.

One should always test that the job script works before submitting it to the grid.

Grid computing

Job jsdl file:

```
<posix:Executable>/bin/sh</..
<posix:Argument>test.sh</..
...
<DataStaging>
<FileName>libs.tar.gz</..
<CreationFlag>overwrite</..
<Source>
<URI>file:///tmp/libs.tar.gz</..
...
<DataStaging>
<FileName>tree.root</..
<DeleteOnTermination>false</..
```

The executable "writing" and all the other files could have been included in one tarball, or all the files could have been sent separately without any tarring.

Assuming a valid grid proxy, how to submit:
`arcsub test.jsdl`
Status checked with `arcstat -a` and result retrieved with `arcget -a`.

File transfers. Simple file transfers can be made using interactive tools, such as gsincftp. File transfers can also be initiated from the client side by arcsub or from the server side by grid-manager as described in the job description.

The session directories are kept on the computing resources for a limited time only, usually at least 24h. Client machines 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.

The job result files can also be transfered into a storage element. Storage elements are persistent storage areas (file servers) which are continuosly connected to the Grid. The storage element can be set by the JSDL command Target

```
<Target>
  <URI>gsiftp://..
```

If you like to move a number of files, best to make a tarball of them, which to transfer.

Grid computing

As an example Storage Element we use dCache server madhatter.csc.fi.

Some useful commands:

ls

- arcls gsiftp://madhatter.csc.fi/pnfs/csc.fi/data/cms/test
- srmls srm://madhatter.csc.fi:8443/pnfs/csc.fi/data/cms/test
- /opt/d-cache/srm/bin/srmls -srm_protocol_version=2 -server_mode=passive -streams_num=1 srm://madhatter.csc.fi:8443/pnfs/csc.fi/data/cms/test

cp

Notice that any nonexistent subdir in the given SE path is automatically created, and wildcards (like star in test.*) are not supported. The output file name must be given explicitly.

- srmcp file://$PWD/test.root srm://madhatter.csc.fi:8443/pnfs/csc.fi/data/cms/test/test.root
- arccp file://$PWD/test.root srm://madhatter.csc.fi:8443/pnfs/csc.fi/data/cms/test/test.root

rm

- srmrm srm://madhatter.csc.fi:8443/pnfs/csc.fi/data/cms/test/test.root
- arcrm srm://madhatter.csc.fi:8443/pnfs/csc.fi/data/cms/test/test.root

ROOT (The data are readable without authentication only from korundi and alcyone)

- TFile* file = new TXNetFile("root://madhatter.csc.fi/pnfs/csc.fi/data/cms/test/test.root");

Grid certificate instructions

Export certificate (Firefox), produces a p12 file:

Preferences→Advanced→Encryption→View Certificates→Your Certificates→Backup

Copy the p12 file to .globus directory (create one if you dont have it).

Create userkey and usercert with openssl, example e.g. in http://ipucu.enderunix.org/view.php?id=2218

```
openssl pkcs12 -nocerts -in mycert.p12 -out userkey.pem
openssl pkcs12 -clcerts -nokeys -in mycert.p12 -out usercert.pem
chmod 0400 userkey.pem
chmod 0600 usercert.pem
```