Practical:
Porting applications to the GILDA grid

Vladimir Dimitrov <vgd@acad.bg>

IPP-BAS

Introduction and basic theory

The main goal:

- **To port and execute an existing non-grid application to the Grid.**

  (In particular, this is GILDA Grid for training purpose.)

Some sources define this process commonly as "**gridifying".** There are many useful and "single-processor" or "single-machine" applications which need gridifying.

Common obstacles

- The candidate-applications for porting usually are huge and complex.
- Some of them use low-level network functions and/or parallel execution features of a specific non-grid environment.
- Usage of non-standard or proprietary communication protocols.
- The complete source code might not be available, might not be well documented or its “out-of-host” usage is restricted by a license agreement.

Common obstacles (continued)

- The application might be written in many different programming languages – C, C++, C#, Java, FORTRAN etc. or even mixture of them.
- Applications may depend on third-party libraries or executables which are not available by default on some Grid worker nodes.
- Some application features could cause unintentional violation of Grid Acceptable Use Policies (Grid AUP).
- Furthermore, the application can have hidden security weakness which will be very dangerous in case of remote Grid job execution.

Common obstacles (continued)

- Some applications are pre-compiled or optimized for using on a machine with particular processor(s) only – Intel, AMD, in 32-bit or 64-bit mode, etc. But the Grid is heterogeneous!
- The application may contain serious  bugs  which have never been  detected while running in a non-grid environment.
- Finally, the formal procedure for accepting a new application to be ported to a Grid for production or even experimental purpose is not simple.

Therefore, the porting of an arbitrary application to Grid could be very long, difficult and expensive process!

The practical

- **Goal: The application called *MatrixDemo* will be ported and executed in GILDA grid environment.** (The program is borrowed from the “EGEE summer school” at MTA SZTAKI institute, Budapest, 2006.)

***MatrixDemo* is written in C programming language.**

Many Grids, especially GILDA and EGEE Grid middleware (gLite) are based on Globus Toolkit, (http://www.globus.org/).

The Globus Toolkit is written in C, so porting the C or C++ programs is easy … probably.

MatrixDemo program

- ***MatrixDemo* program performs some matrix operations – inverting, multiplying, etc.**

- **Usage:**
- ***MatrixDemo* has command line interface which accepts several arguments. Starting the program without any argument will display a short help.**
  - Example:

```
MatrixDemo I V
```

This will Invert (`I`) the matrix defined in the file named INPUT1 and will store the result in the file OUTPUT with verbose details (`V`).

MatrixDemo program (continued)

- **Prerequisites:**

✓ **File *MatrixDemo.c*** – the source code of the program.

✓ **Files *INPUT1* and *INPUT2*** – they contain matrix data in the following text format:

rows, columns, cell1, cell2, cell3 …

Where *rows* is an integer representing the number of rows. *columns* represents number of columns, and *cell1, cell2* etc. are the cells of the matrix, floating point numbers separated by commas (,).

✓ **A standard C compiler and linker.** In this case we will use GNU C (gcc) already installed.

✓ **File *MatrixDemo.jdl*** – a prepared JDL (Job Description Language) file.

MatrixDemo: compilation

- **Step:**

1. **Log on to the GILDA user interface using PuTTY client located on your Windows Desktop.** (The user input is given in red color.)

   Hostname: **`glite-tutor.ct.infn.it`**

   login as: **`sofiaXX`**

   (where *XX* must be 01, 02 etc.)

   Password: **`GridSOFXX`**

   (where *XX* is the same number as above)

   For example: the password for user **sofia15** is **GridSOF15**.

MatrixDemo: getting the prerequisites

- **Step:**

2. **Download the prerequisites stored in a zipped file *MatrixDemo.zip* with the following command:**

```
wget http://vgd.acad.bg/MatrixDemo.zip
```

Unzip the archive in your current directory with the command:

```
unzip MatrixDemo.zip
```

(This will create a subdirectory *matrix* with all of the prerequisite files inside.)

Change the current directory:

```
cd matrix
```

MatrixDemo: compilation

- **Step:**

3. **Compile and link the program using GNU C compiler / linker:**

```
gcc -o MatrixDemo MatrixDemo.c
```

This will create an executable file *MatrixDemo*.

Look at the directory contents:

```
ls -l
```

MatrixDemo: testing as a non-grid application

- **Step:**

4. **Invert the matrix stored in *INPUT1* file with the following command:**

```
./MatrixDemo I V
```

Look at the content of the input file *INPUT1*:

```
more INPUT1
```

Look at the content of the output file *OUTPUT*:

```
more OUTPUT
```

And you may examine the source code:

```
more MatrixDemo.c
```

MatrixDemo: preparing the corresponding JDL-file

- **Step:**

5. **Look to the supplied *MatrixDemo.jdl* file:**

```
more MatrixDemo.jdl
```

The *MatrixDemo.jdl* contents:

```
[
        VirtualOrganisation = "gilda";
        Executable = "MatrixDemo";
        JobType = "Normal";
        Arguments = "I V";
        StdOutput = "std.out";
        StdError = "std.err";
        InputSandbox = {
                "MatrixDemo",
                "INPUT1"
                };
        OutputSandbox = {
                "std.out",
                "std.err",
                "OUTPUT"
        }
]
```

MatrixDemo: preparing the JDL-file (continued)

- **Short explanation of some important JDL-attributes:**
  - **`VirtualOrganisation`** – this points to our training VO (gilda);
  - **`Executable`** – sets the name of the executable file;
  - **`Arguments`** – command line arguments of the program;
  - **`StdOutput, StdError`** - files for storing the standard output and error messages output;
  - **`InputSandbox`** – input files needed by the program, including the executable;
  - **`OutputSandbox`** – output files which will be written during the execution, including standard output and standard error output;

gLite: entering the Grid!

- **Step:**

6. **Authenticate yourself as a Grid user with gilda VO membership:**

```
voms-proxy-init  --voms gilda
```

This will ask for the passphrase which is SOFIA for all users.

Check the proxy status with:

```
voms-proxy-info
```

gLite: Checking the job requirements

- **Step:**

6. Investigate the abilities to run the job among the Grid-sites with gilda VO support:

```
glite-job-list-match MatrixDemo.jdl
```

This command will produce a listing with all of the Grid Computing elements together with jobmanager queues that fulfill the requirements of our job.

gLite: Submitting the job to GILDA Grid

- **Steps:**

7. **Execute the following command:**

```
glite-job-submit -o MatrixDemo.id MatrixDemo.jdl
```

This will submit the job and will store its unique identifier in a file called *MatrixDemo.id.* You may look at that file.

8. **Monitor the job status with:**

```
glite-job-status -i MatrixDemo.id
```

Execute this command several times until **“Done (Success)”** status.

gLite: retrieving the job results

- **Step:**

9. **Execute the following command:**

```
glite-job-output –i MatrixDemo.id –dir ./
```

This will retrieve the Output sandbox files and will store them into a local directory with a strange name under the current directory. Directory name will be something like *sofia01_aJiesiAtu96H09XASy_j_Q.*

Enter the output directory and look at the files named *OUTPUT* and *std.out*

```
more OUTPUT
more std.out
```

Exercise!

- **Modify and execute the job in a way that it will produce multiplication of the two matrices stored in *INPUT1* and *INPUT2* files. Check up the result by hand. ;-)**

Questions?