Gridification practices in gLite 3.x

Vladimir Dimitrov
IPP-BAS

"gLite middleware Application Developers Course",
Sofia, Bulgaria, 25.11.2008

Outline

- **This talk gives a high-level view of application development in Grids**
- **Contents**
  - Review of concepts: grids and grid applications
  - Characteristics of VOs
  - Challenges to researchers who write applications
  - General steps of application gridification
  - Practical: Preparing and submitting a job based on a simple non-grid application.
- **Acknowledgements**
  - Gergely Sipos, SZTAKI, Hungary
  - Miklos Kozlovszky, SZTAKI, Hungary
  - Mike Mineter, University of Edinburgh, UK, “Application Development and Aspects of gLite 3.0”, 2006
  - Douglas Thain, The University of Notre Dame
  - GILDA team

What is a grid application?

- ***Definition***

Software that interacts with Grid services to achieve requirements that are specific to a particular Virtual Oraganization (VO) or user.

Characteristics of VOs

- **What is being shared?**
  - resources of storage and/or compute cycles
  - software and/or data

- **Distinct groups of developers and of users?**
  - Some VOs have distinct groups of developers and users…
    - Biomedical applications used by clinicians,….
  - …. Some don't
    - Physics application developers who share data but write own analyses
  - Effect: need to
    - hide complexity from the 1st type of VOs
    - expose functionality to 2nd type of VOs
    - **many security issues**

Basic tasks while Porting applications to the Grid

1. **Developing a non-Grid application (or inheriting and adapting an ancient one);**
2. **Go/no-Go decision about gridification**
   - Is it suitable for the Grid environment?
   - “Cost/profit” analysis / feasibility study

Typical Questions Groups:

- Current structure of the application
- Dependencies of the application
- Available resources (manpower, knowledge, etc.)
- Requirements for the gridified application
- Expected impact of gridification
- Requirements for the grid infrastructure

Basic tasks while Porting applications to the Grid (contd.)

3. **Grid environment access**
   - Requesting Certificates / VO membership
   - Accessing Grid environment
     - Appropriate VO **UI** machine account for command line
     - Portal GUI account;

4. **Executing, Testing and Debugging the application;**
   - Testing the non-grid application (debugging in Grid environment is a hard task), creating use cases for single (non-grid) runs;

5. **Constructing the job suite – JDL (Job Description Language) files, executables, auxiliary scripts and input/output data files;**

6. **Submitting the job to the Grid as a small-scale pilot application;**

Basic tasks while Porting applications to the Grid (contd.)

7. **Executing, Testing and Debugging the pilot application;**

8. ***IF* something goes wrong**

   THEN GOTO 4;

9. ***IF* everything seems to work**

   THEN increase the scale of the application (increase problem size, amount of used resources);

10. **Optimizing and improving the Grid application;**

Porting legacy code applications

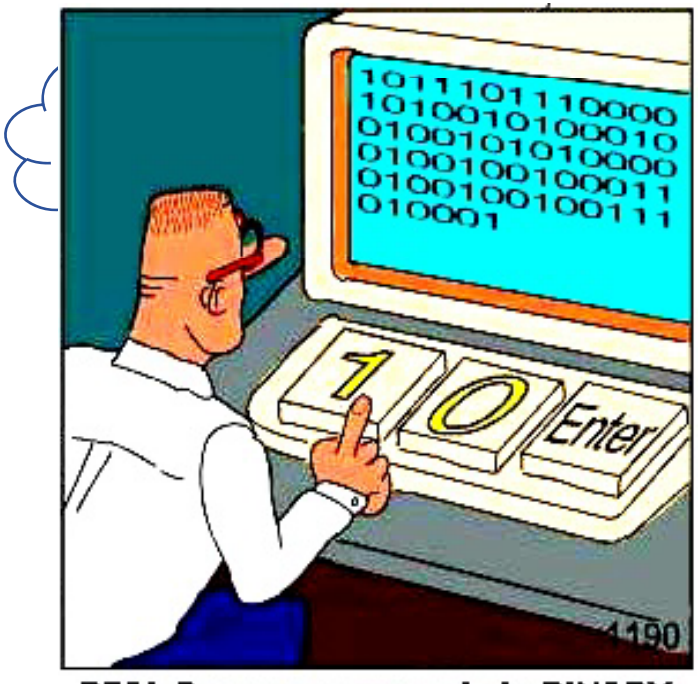
Code from the past, maintained because it works

Often supports business critical functions

Not Grid enabled

What to do with legacy codes in service Grids?

- Bin them and reimplement them as grid services
- Reengineer them → who knows the source code?
- **Port them onto the Grid with minimum user effort!**



Practical tools @ gridification

Hiding Grid remote storage system from a legacy application

- **Parrot**
  - a handy tool for attaching old programs to new storage systems
  - EGEE (gLite module) Data Access: GFAL, LFN, GUID, SRM, RFIO, DCAP, and LFC
  - does not require any special privileges, any recompiling to existing programs
  - For example, an anonymous FTP service is made available to **vi** like so:

```
parrot vi /anonftp/ftp.cs.wisc.edu/RoadMap
```

Or example with gsiftp:

```
$./parrot app_name /gsiftp/<gsiftp usr without gsiftp://>
```

More info:

http://www.cse.nd.edu/~ccl/software/manuals/parrot.html

Grid Application Development Support Model

- **8 steps support model**
  - Contact phase
  - Pre-selection phase
  - Analysis phase
  - Planning phase
  - Prototyping phase
  - Testing phase
  - Execution phase
  - Dissemination and feedback phase

In the practical

- **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, 2006.)

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

GILDA environment (gLite based) is supporting C,
so porting the C or C++ programs is easy … hopefully.

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.

Log on to User Interface

- **Step:**

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

   Hostname: **`ui.grid.acad.bg`**

   login as: **`sofia`*`XX`***

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

   Password: **`GridSOF`*`XX`***

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

   For example: the password for user **sofia12** is **GridSOF12**.

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/grid/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 and JDL file:

```
more MatrixDemo.c
more MatrixDemo.jdl
```

gLite: entering the Grid!

- **Step:**

5. **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-wms-job-list-match -a 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-wms-job-submit -a -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-wms-job-status -i MatrixDemo.id
```

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

Common problems and obstacles

Now some bad news:

- 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 a 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.
- Lack of convenient Grid-enabled program development tools, debuggers and profilers.
- Finally, the formal procedure for accepting a new application to be ported to a Grid for production or even for experimental purpose is not simple.

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

Back to *MatrixDemo*

- **Step:**

9. Execute the following command:

```
glite-wms-job-output -i MatrixDemo.id --dir out
```

This will retrieve the Output sandbox files and will store them into a local directory named **out**.

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

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

Questions?