Automating the grid work

Riccardo Di Meo

Working with the grid is easy, but...

- Tough single operations aren't complicated, they are usually reiterated very frequently becoming:
  - boring
  - error prone
  - inefficient
- Portals and workflows solve those issues, although their's static user interface isn't customizable as a plain shell approach.

CLI approach

Pro

- More customizable.
- Full control over the grid (as far as the architecture allows).
- Easier than the portal approach in *some* cases.
- People who consider programming an art will love it...

Cons

- Overkill in many situations.
- Need more work to be carried out.
- More difficult than the portal approach in *many* cases.
- People who consider programming a tool will hate it...

What should a CLI suite provide?

- Every step from submission to result retrieval should be carried out automatically.
- Failure in the job execution should be taken into account and handled appropriately.
- data retrieval should include application specific steps which process the results into a manageable/useful form.

What tools should be used?

- A scripting language has to be chosen, like:
  - Bash
    - tough powerful and present in almost every linux installation, does not provide all the tools needed to build useful scripts, which will always depend from external programs.
  - Python
    - complete and easy to learn, Python is a de facto standard component in every Linux distribution.
  - PERL
    - probably the most used scripting language: tough powerful has a tricky syntax which make it not very easy for newbies.
  - TCL/Tk
    - not wide spreed as the previous ones, is easy to learn and provides powerful facilities to build nice looking GUI, although like Bash relies on external programs.
- A minimum knowledge of the shell used is necessary.

Tools that we will use.

- Since BASH is the easiest approach for simple tasks and a basic working knowledge of it is necessary to work with computers, we will use it for this tutorial.
- A basic knowledge about it will be required, precisely:
  - copy/move/edit files.
  - pipe the result of a command to another one.
  - save data into a variable.
  - edit a text file with a program of your choice.

Some simple constructs in bash

- variable assignments:
  - `a=10`
  - `b=`ls``
- substitutions:
  - `echo $a` (will print 10)
  - `echo “$a” (again, 10)`
  - `echo '$a' (will print $a)`
- piping:
  - To a file:
    - `echo $a >test.txt` (create a file and put the value 10 in it)
  - To the end of it:
    - `echo $a >> test.txt` (if test.txt is present, append 10 to it, else behaves as the former command)
  - Through another command:
    - `echo 'plot sin(x) w l' | gnuplot -persist`

More advanced constructs

- "For" cycles:

```
for((i=0;i<10;i++))
    do
    echo -n $i
done
```

  - and

```
for i in 0 1 2 3 4 5 6 7 8 9
    do
    echo -n $i
done
```

  - which produce:

```
0123456789
```

which can be used to repeat a command for a set number of times (e.g. for every file with a specified extension in a directory)

As already mentioned, other programs are required in order to create a (useful) script in bash: we will describe them along the way.

Some commands are already implemented in bash, the remaining ones can be found in a standard Linux distribution.

A good (basic) guide can be found here:

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/index.html

and an advanced one here:

http://www.tldp.org/LDP/abs/html/index.html

documentation for (almost) every command we will use can be found in the man pages (typing man *command name*) or in the info pages (info *command name*).

A simple bash script

```
#!/bin/bash
# Lines starting with '#' are comments: save this program in a file
# called “fibo.sh”. We will use this script as test program for the
# next examples.
a=$1
b=$2
end=$3
for((i=0;i<end;i++))
    do
    let delay=3*$RANDOM/32767
    sleep $delay
    echo $i” “$a
    let c=$a+$b
    a=$b
    b=$c
done
```

Handling the JDLs

- As you probably know, JDL are used to specify a grid task.
- Often you are required to launch multiple copy of a JDL with very small variations (changed file name, parameters ecc...).
- Doing this by hand is boring and time consuming.
- With bash is easy and straightforward. As long as you know how ;-) : the basic idea is to create a JDL template and to use bash to create the single variations by substituting some parameters in it.

Example

Let's assume that fibo.sh is the program we will need to put on the grid and that the output is the result we want to collect for different input parameters: below you see the JDL for a single call (fibo1.jdl):

```
[
    Executable = "fibo.sh";
    Arguments = "1 1 10";
    StdOutput  = "result.txt";
    StdError ="stderr.txt";
    InputSandbox = {"fibo.sh"};
    OutputSandbox = {"result.txt","stderr.txt"};
]
```

- Assume that fibo.sh were a very complicated program (we don't have time to make on on the fly...) that produce useful and nontrivial results for a change in the parameters and you will get the picture...

Dumb way of working

- Assume you want to “explore” the results of fibo in respect of the first 2 parameters in a range of [0:10] for and [0:20] respectively, leaving the third to “10” you should:
  - open fibo.jdl.
  - modify the parameters in the “Arguments” field
  - save the file
  - submit it
- and all that should be done about 200 times (or more, if some simulation fail!), and this is a **very simple** example!

Clearly we need a faster approach!

The smart way

- Create this file and save it in "*fibo_jdl.template*":

[

Executable = "fibo.sh";

Arguments = "FIRST SECOND THIRD";

StdOutput = "result.txt";

StdError ="stderr.txt";

InputSandbox = {"fibo.sh"};

OutputSandbox = {"result.txt","stderr.txt"};

]

- Now try this command:

```
cat fibo_jdl.template | sed  \
  "s/FIRST/10/;s/SECOND/20/;s/THIRD/10/"
```

- "**sed**" is a utility that (among other things) can be used to substitute text in a file.

Let's put those things together!

```
#!/bin/bash
# Call this file “simple_submit.sh”
# Read the parameters from the user
echo -n “Max a? “
read amax
echo -n “Max b?”
read bmax

# For every parameter selection, substitute the values in a file called
# fibo.jdl and submit it. Save the id in a file called id_list.txt

for((i=0;i<=$amax;i++))
 do
  for((j=0;j<=$bmax;j++))
   do
   echo “substituting $i and $j as first and second parameters...”
   cat fibo_jdl.template|sed "s/FIRST/$i/;s/SECOND/$j/;s/THIRD/10/" >fibo.jdl
   echo “Submitting the resulting jdl...”
   echo 'edg-job-submit  -o id_list.txt fibo.jdl' |tee >> id_list.txt
   done
  done
```

Some notes...

- There's a “echo” command in the line of the edg-job-submit to prevent the submission (the id_list is bogus). This is a standard procedure while working on/debugging a script.
- Try now to modify the previous script in order to:
  - ask the starting point for the first and second parameters and correct the for cycles.
  - ask the user for the third parameter and substitute it in the template (you can do it in 2 way: the most easy is to assume that the param. doesn't change among simulations, the other is to build another cycle, thus allowing the user to explore a 3 dimensional parameter space).
  - Save all the generated fibo.jdl files in another directory in order to allow the user to inspect them during/after the submission.