RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Hands-On with BigJob

Presented by: Pradeep Kumar Mantha
The RADICAL Group
http://radical.rutgers.edu
http://saga-project.org

Agenda

- Introduction
- BigJob Components
- Pilot Description
- Compute Unit Description
- Install BigJob
- Hands-On Simple Ensembles
- Hands-On Mandelbrot
- Hands-On Chained Ensembles
- Hands-On Coupled Ensembles
- Hands-On Scaling Jobs
- Hands-On Compute Data

Introduction

- What is BigJob?
  - A SAGA-based Pilot-Job Framework
  - Lightweight Python scripts utilizing SAGA BlisS' capabilities
- BigJob allows for the execution of multiple jobs without the necessity to queue each job.
- BigJob allows user-level, programmable control of the execution environment and construction of complex workflows
- Decouples the task coordination from the task execution
- Provides the logic of the execution without having to schedule the tasks individually

Please Point Your Browser…

- Follow along with this tutorial by visiting….

XSEDE Tutorial Part 3: BigJob

https://github.com/saga-project/BigJob/wiki/XSEDE-Tutorial-Part-3%3A-BigJob

BigJob Components

- There are two main components of concern when writing BigJob scripts.
  - **Pilot Description**: Defines the resource specification for managing jobs on each resource
  - **Compute Unit Description**: Defines the actual job description and data for movement

- Quick note: This set of slides assumes that you have followed part 3 of this tutorial and that you have installed BlisS and are logged in via ssh to the Lonestar XSEDE resource.

```
cd $HOME
```

Pilot Description

- Resource specification requirements
  - **service_url:** Specifies the SAGA Bliss job adaptor and resource hostname on which jobs can be executed. For remote hosts, password-less login must be enabled.
  - **number_of_processes:** specifies the total number of processes that need to be allocated to run the jobs
  - **queue:** Specifies the name of the job queue to be used
  - **working_directory:** specifies the directory in which the Pilot-Job agent executes
  - **walltime:** Specifies the number of minutes that the resources are requested
  - **file_transfers:** specifies the files that need to be transferred in order to execute the jobs successfully. Generally files common to all jobs are listed here.

Pilot Description Code

- What does it all look like in Python?

```
pilot_compute_description.append({ "service_url": "sge+ssh://
localhost",
"number_of_processes": 12,
"allocation": "XSEDE12-SAGA",
"queue" : "development",
"working_directory": os.getenv( "HOME")+ "/agent",
"walltime":10 })
```

Compute Unit Description

- CU Requirements (job description, etc):
  - **executable:** The code or software that you are trying to run/execute
  - **arguments:** The list of arguments that will be passed to the executable (i.e. command line arguments)
  - **environment:** specifies the list of environment variables to be set for successful job execution
  - **working_directory:** The directory in which the job has to execute. If left unspecified, the Pilot-Job creates a default directory
  - **number_of_processes:** specifies the number of processes to be assigned for the job execution
  - **spmd_variation:** specifies the type of job. By default, it is single job.
  - **output:** The file in which the standard output of the job execution will be stored
  - **error:** The file in which the standard error of the job execution will be stored (will be a blank file if no errors)
  - **file_transfers:** The files that need to be transferred in order to execute the job successfully.

Compute Unit Description Code

- What does it all look like in Python?

```
compute_unit_description = { "executable": "/bin/echo",
"arguments": ["Hello","$ENV1","$ENV2"],
"environment": ['ENV1=env_arg1','ENV2=env_arg2'],
"number_of_processes": 4,
"spmd_variation":"mpi",
"output": "stdout.txt",
"error": "stderr.txt" }
```

Install BigJob

- BigJob is available via PyPi and can be installed via pip.

```
pip install bigjob
```

- Verify that your installation was successful

```
python -c "import bigjob; print bigjob.version"
```

- Create BigJob agent directory. BigJob uses this agent directory as it's 'working directory' in our examples.

```
mkdir $HOME/agent
```

Hands-On: Simple Ensembles

- The first example submits N jobs using BigJob. It demonstrates the mapping of a simple /bin/echo job.
- Verify that you are in your home directory

```
cd $HOME
```

- Open a new file named *simple_ensembles.py* in your favorite text editor (vim, emacs, etc)

```
vim simple_ensembles.py
```

- Copy and paste the contents from the website into your file and save it.

Hands-On: Simple Ensembles, cont'd

- Execute the script

```
python simple_ensembles.py
```

- How do you check the output from your script?

```
cd agent
cd bj-UID
cd sj-UID
cat stdout.txt
```

Hands-On: Mandelbrot

- To demonstrate a more complex workflow example, we will show the Mandelbrot example from .
- First, verify that you have followed through the second part of the tutorial and downloaded the python image library and the mandelbrot.py script.
- Open a new file named *bj_mandelbrot.py* in your favorite text editor (vim, emacs, etc)

```
vim bj_mandelbrot.py
```

- Copy and paste the contents from the website into your file and save it.

Hands-On: Mandelbrot, cont'd

- Execute the script

```
python bj_mandelbrot.py
```

- View your output in the agent directory

How does the execution of bj_mandelbrot.py differ from saga_mandelbrot.py (part 2)?

Hint: Try to find the execution time difference via SAGA Bliss Mandelbrot and Pilot-Job Mandelbrot. Why is the time difference?

Hands-On: Chained Ensembles

- Submits a set of echo jobs(A) using SAGA Pilot-Job, and for every successful job (with state Done), it will submit another /bin/echo job (set B) to the same Pilot-Job. Open a file and copy-paste the contents

```
vim chained_ensemble.py
```

```
while 1:
        for i in all_A_cus:
            if i.get_state() == "Done":
                compute_unit_description = { "executable": "/bin/echo",
                                             "arguments": ["$ENV1","$ENV2"],
                                             "environment":
['ENV1=task_B:','ENV2=after_task_A'+str(i)],
                                             "number_of_processes": 1,
                                             "output": "B_stdout.txt",
                                             "error": "B_stderr.txt"
                                           }
                compute_data_service.submit_compute_unit(compute_unit_description)
                all_A_cus.remove(i)

        if len(all_A_cus) == 0:
            break
```

Hands-On: Chained Ensembles, cont'd

- Execute the script

```
python chained_ensemble.py
```

- View your output in the agent directory.

Can you identify what job of set B executed after job of set A?

```
cd agent
cd bj-UID
cd sj-UID
cat stdout.txt
```

Hands-On: Coupled Ensembles

- Submits a set of jobs(A) and jobs(B) and wait until they are completed and then submits set of jobs(C). It demonstrates synchronization mechanisms provided by SAGA Pilot-API..
- Verify that you are in your home directory

```
cd $HOME
```

- Open a new file named *coupled_ensembles.py* in your favorite text editor (vim, emacs, etc)

```
vim coupled_ensembles.py
```

- Copy and paste the contents from the website into your file and save it.

Hands-On: Coupled Ensembles, cont'd

- Execute the script

```
python coupled_ensemble.py
```

- View your output in the agent directory.

Can you identify what job of set C executed after job of sets A and B?

```
cd agent
cd bj-UID
cd sj-UID
cat stdout.txt
```

Hands-On: Scaling Jobs

- Provides an example to submit jobs to multiple pilots. The Pilot-API manages the jobs across multiple pilots launched on same/different machines.
- Verify that you are in your home directory

```
cd $HOME
```

- Open a new file named *scaling_jobs.py* in your favorite text editor (vim, emacs, etc)

```
vim scaling_jobs.py
```

- Copy and paste the contents from the website into your file and save it.

Hands-On: Scaling Jobs, cont'd

- Execute the script

```
python scaling_jobs.py
```

- Notice how the script can be extended for multiple resources and different batch queuing systems
- View your output in the agent directory.

```
cd agent
cd bj-UID
cd sj-UID
cat stdout.txt
```

Hands-On: Compute-Data

- Demonstrates associating data required for the successful execution of executable. The Pilot-API is responsible for moving the necessary data to the executable working directory.
- Verify that you are in your home directory

```
cd $HOME
```

- Open a new file named *compute_data.py* in your favorite text editor (vim, emacs, etc)

```
vim compute_data.py
```

- Copy and paste the contents from the website into your file and save it.

Hands-On: Compute-Data, cont'd

- Create a test.txt file in $HOME directory. This will serve as the file that is moved around with the executable.

```
cat /etc/motd > $HOME/test.txt
```

- Execute the python script.

```
python compute_data.py
```

- Look in the subjob directory for the file, along with stdout.txt and stderr.txt

```
cd agent
cd bj-UID
cd sj-UID
cat stdout.txt
cat test.txt
```

Conclusion

- BigJob supports remote job submission
- Decouples the task coordination from the task execution
- Allows the execution of jobs without the necessity to queue each individual job hence reducing time to soluton
- Allows you to effectively utilize resources wherever they become available, e.g. XSEDE, OSG, FutureGrid… “scale-across”

BigJob Support

- bigjob-users@googlegroups.com

Questions?