JASMIN Workshop: Exercise 03: Processing on the scientific analysis servers

Scenario

I need to do a random sampling of my dataset to estimate the distribution of sample mean. I need to use the random number generator function available in the numpy library in Python. I need to test this function by running a standalone Python script on the scientific analysis servers prior to running the statistical analysis script.

Note: If needed refer to the cheat sheet for exercise 3 at the end of this document for specific commands (example of a command output is in blue text color)

Objectives

- To be aware of the capabilities and limitations of the scientific analysis servers
- To be able to test/run a script/code on them
- To be able to monitor your interactive processes and share resources with other users

JASMIN resources

- Scientific analysis servers
- Group workspace: `/group_workspaces/jasmin2/workshop/users/$USER/ex03`
- A Python example script is provided: `/group_workspaces/jasmin2/workshop/exercises/ex03/src/random-number-gen.py`

Local resources

- SSH client (to login to JASMIN)

Instructions

1. Launch two terminals. Start ssh-agent session and add JASMIN private key to login to JASMIN on each terminal
2. SSH to the same scientific analysis server on each terminal
3. On the execution terminal, copy the Python example script (shown in the JASMIN resources section) to your current working directory
4. On the monitoring terminal, launch the Linux command `top -u <username>` to monitor your processes on the scientific server. **Note**: to check the load on JASMIN sci servers use the command `top` to sort all processes per CPU usage and `top -a` to sort by memory usage
5. On the execution terminal, run the Python example script using the command `python2.7 random-number-gen.py`
6. On the monitoring terminal, check the Python example script process ID (pid), state, memory and CPU usage
7. On the monitoring terminal, press the keyboard letter `q` to exit the `top` command
8. Check if the script spawns threads using the Linux command: `ps -T -p <pid>`
9. On the execution terminal, edit the Python example script `random-number-gen.py` and decrease the size of the random numbers from 1024 to 500
10. On the execution terminal, rerun the Python example script using the command `python2.7 random-number-gen.py`

11. On the monitoring terminal, check the Python example script process ID (pid), state, memory and CPU usage for generating 500 random numbers
12. Logout from the scientific server and from the login server

Review

You will be able to run a Python script on the scientific analysis servers. You will be able to monitor the resources used by your script on the scientific analysis servers. You can scale up by using the high-memory scientific jasmin-sci[3,6] server for a large set of random numbers.

Alternative approaches and best practice

- Do not run parallel applications e.g. MPI or OpenMP, high threaded codes on the scientific analysis servers
- Do not run data transfer processes on the scientific analysis servers. Please use `jasmin-xfer1` (Except when moving data from `/work/scratch` to a GWS because `/work/scratch` is not mounted on `jasmin-xfer1`)
- Do not generate huge numbers of files (>1000) in a single directory
- Do not use `/tmp` on the scientific servers and transfer servers. Using `/tmp` can cause the scientific analysis server to crash, resulting in loss of work. Set the environment variable `TMPDIR` to a temporary directory under a GWS area- `export TMPDIR=/GWS-path/<your_project>/<your_username>/tmp`
- Do not run processes with execution time over two hours
- Limit the number of threads when testing a multithreaded code on scientific analysis servers
- Use the high memory scientific servers `jasmin-sci[3,6]` for processes with memory usage over 1 GB and/or threads.
- It is necessary to consider moving processing to the batch system when the resources demand is high.
- Many instances of e.g. Ipython or IDL can impact the performance of the scientific servers. Please note that for IDL, we have a large pool of run-time licences and a much more limited pool of development licences.
- Use the high memory scientific analysis servers `jasmin-sci[3,6]`for testing high memory and/or multithreaded code (`jasmin-sci3` (48 CPUs, 2117GB RAM), `jasmin-sci6` (24 CPUs, 384GB RAM))

https://help.jasmin.ac.uk/article/121-sci-servers
https://help.jasmin.ac.uk/article/176-storage

Cheat sheet for Exercise 03: Processing on the scientific analysis servers

1. Start the ssh-agent and load your SSH private key. Login to JASMIN - launch two login sessions

```
exec ssh-agent $SHELL
ssh-add ~/.ssh/id_rsa_jasmin
ssh -A <username>@jasmin-login1.ceda.ac.uk
```

2. SSH login to a scientific analysis server from the two login sessions, one for execution and one for monitoring

```
ssh jasmin-sci<number>.ceda.ac.uk
```

3. On the execution terminal, copy the Python script `random-number-gen.py` that generates random numbers to your current working directory

```
cp /group_workspaces/jasmin2/workshop/exercises/ex03/src/random-number-gen.py .
```

4. On the monitoring terminal, monitor your processes by executing the Linux command. Note that you have two `sshd` processes referring to your two SSH login sessions.

```
top -u <username>
top - 17:18:36 up 19 days,  7:39, 34 users,  load average: 30.69, 30.60, 30.79
Tasks: 960 total,   2 running, 951 sleeping,   7 stopped,   0 zombie
Cpu(s):  3.2%us,  1.3%sy,  0.0%ni, 95.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  395365628k total, 191724156k used, 203641472k free,   453400k buffers
Swap:  4194300k total,  1407916k used,  2786384k free, 76732356k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
426965 freddy    20   0 13976 1912  904 R  0.7  0.0   0:00.08 top
426313 freddy    20   0  112m 1944  852 S  0.0  0.0   0:00.00 sshd
426314 freddy    20   0  104m 1840 1380 S  0.0  0.0   0:00.01 bash
426390 freddy    20   0  112m 1948  852 S  0.0  0.0   0:00.00 sshd
426391 freddy    20   0  104m 1820 1372 S  0.0  0.0   0:00.01 bash
```

5. On the execution terminal, execute the Python script: `python2.7 random-number-gen.py`

```
python2.7 random-number-gen.py
1024  ======>>> random numbers
I am sleeping for 40 seconds so you can check the resources usage
```

6. On the monitoring terminal, check the memory and CPU usage of the Python script when it is in a running state `R`?

```
top -u <username>
   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
427708 freddy    20   0 47.6g 699m 5948 R 125.5  0.2   0:04.17 python2.7
427619 freddy    20   0 13976 1912  904 R  1.0  0.0   0:00.10 top
426313 freddy    20   0  112m 1944  852 S  0.0  0.0   0:00.01 sshd
426314 freddy    20   0  104m 1840 1380 S  0.0  0.0   0:00.01 bash
426390 freddy    20   0  112m 1948  852 S  0.0  0.0   0:00.00 sshd
426391 freddy    20   0  104m 1832 1376 S  0.0  0.0   0:00.01 bash
```

7. On the monitoring terminal, check the memory usage of the Python script when it is in a sleeping state `S`. Then, press the keyboard letter `q` to exit the `top` command

```
   PID USER      PR  NI  VIRT  RES   SHR S %CPU %MEM    TIME+  COMMAND
428295 freddy    20   0 13972 1912   904 R  0.7  0.0   0:00.18 top
426313 freddy    20   0  112m 1952   852 S  0.0  0.0   0:00.06 sshd
426314 freddy    20   0  104m 1840  1380 S  0.0  0.0   0:00.06 bash
426390 freddy    20   0  112m 1948   852 S  0.0  0.0   0:00.04 sshd
426391 freddy    20   0  104m 1832  1376 S  0.0  0.0   0:00.02 bash
427708 freddy    20   0 48.0g 1.0g  5948 S  0.0  0.3   0:04.39 python2.7
```

8. On the monitoring terminal, check if the Python example script spawned threads

```
ps -T -p 427708
427708 427708 pts/86   00:00:01 python2.7
```

9. On the execution terminal, launch a text editor to edit the Python example script `random-number-gen.py`. Decrease the size of the random numbers from 1024 to 500 on the line of code starting "`nran =`"

```
import numpy as np
import time

# Number of random numbers to be generated
nran = 1024

# Generate a random number from the normal distribution
result = [np.random.bytes(nran*nran) for x in range(nran)]

print(len(result), "======>>> random numbers")

# Wait for tsleep seconds
tsleep = 40

print("I am sleeping for {0} seconds so you can check the resources
usage".format(tsleep))
time.sleep(tsleep)
```

10. On the execution terminal, re-run the edited Python example script

```
python2.7 random-number-gen.py
500 ======>>> random numbers
I am sleeping for 40 seconds so you can check the resources usage
```

11. On the monitoring terminal, check the Python example script process ID (pid), state, memory and CPU usage for generating 500 random numbers

```
top -u <username>
   PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
103508 freddy     20   0  550m  37m 5948 R 32.5  0.1   0:00.99 python2.7
103369 freddy     20   0 13692 1856 1180 R  1.3  0.0   0:00.33 top
104330 freddy     20   0  111m 1512  420 S  0.0  0.0   0:00.00 sshd
104331 freddy     20   0  104m  744  276 S  0.0  0.0   0:00.05 bash
110882 freddy     20   0  111m 1564  452 S  0.0  0.0   0:00.04 sshd
110883 freddy     20   0  104m 1352  884 S  0.0  0.0   0:00.08 bash
```

12. Logout

```
Logout
Connection to jasmin-sci<number>.ceda.ac.uk closed.
```