

Efficient Submission of Serial Jobs

Aaron Holt

Slides: https://github.com/ResearchComputing/Final_Tutorials

Tools

1) Bash scripting

2) GNU Parallel

3) Loadbalancer

Batch Script with One Serial Task

```
#!/bin/bash
#SBATCH --partition janus
#SBATCH --job-name process_file
#SBATCH --nodes 1
#SBATCH --output process_file.out
#SBATCH --time 01:30:00

python main.py input_file_1.csv
```

Batch Script with Multiple Tasks

Serial Scripts on One Processor	Serial Scripts on Multiple Processors
<pre>#!/bin/bash #SBATCH --partition janus #SBATCH --nodes 1 #SBATCH --time 18:00:00 python main.py input_file_1.csv python main.py input_file_2.csv ... python main.py input_file_12.csv</pre>	<pre>#!/bin/bash #SBATCH --partition janus #SBATCH --nodes 1 #SBATCH --time 01:30:00 python main.py input_file_1.csv & python main.py input_file_2.csv & ... python main.py input_file_12.csv & wait</pre>

Your Turn

1) In a terminal, background 2
'sleep' commands which pause
for 30 seconds. Use 'wait' to wait
for the commands to finish

- You may use a login node or
a personal computer (Need a
Bash terminal)

Useful Commands	
&	This will background a process when placed at the end of a command
sleep N	Pause for N seconds
wait	Wait for a process state change

Login:

- ssh username@rc.colorado.edu
- ssh user00xx@tutorial-login.rc.colorado.edu

Solutions

On separate lines:

sleep 30 &

sleep 30 &

wait

On one line:

sleep 30 & sleep 30 & wait

Your Turn

Submit a job to Janus

Setup

- Login to Janus
 - `ssh username@login.rc.colorado.edu`
 - `ssh user00xx@tutorial-login.rc.colorado.edu`
- Load slurm
 - `module load slurm`

Your Turn

2) Submit a slurm job to Janus which backgrounds 6 'sleep' commands which pause for 10 seconds.

- Batch filename called sleep.sh
- Commands should run in parallel!
- Use 1 node in default queue
- 2 minute wall time
- Output file called sleep.out
- Reservation scbasics2

Useful Commands

&	Background a process when placed at the end of a command
sleep N	Pause for N seconds
wait	Wait for a process state change
sbatch file.sh	Submit file.sh to slurm (need to load slurm with 'ml slurm' first)

Sbatch Options

--nodes N	Request N nodes
--output	Specify an output filename
--reservation	Specify reservation to run in
--time	Wall time, format HH:MM:SS

Solution

sleep.sh

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --time 0:02:00
#SBATCH --output sleep.out
#SBATCH --reservation scbasics2
sleep 10 &
sleep 10 &
sleep 10 &
sleep 10 &
sleep 10 &
sleep 10 &
wait
```

Submit with:

sbatch sleep.sh

Bash Script Summary

- You don't need a special tool
- Available almost everywhere
- Takes some bash experience to write more complex scripts
- Not great for running large numbers of tasks

GNU Parallel

- What if you want to run more than 12 tasks on a node? More than 100 tasks? More than 1000?
- The bash script may cause inefficiencies or errors if many tasks are started all at once.
- GNU parallel is a shell tool for executing tasks in parallel using one or more computers.
 - In it's simplest form, GNU parallel is a parallel replacement of a for loop.
- Options to specify how many tasks should run in parallel, display output in order, limit resources and more!

GNU Parallel Examples

parallel [options] [*command* [arguments]] (::: arguments | :::+ arguments | ::: argfile(s) | :::+ argfile(s)) ...

Two ways of printing numbers 1 to 4 in parallel:

```
parallel echo {} ::: 1 2 3 4
```

```
seq 1 4 | parallel echo {}
```

GNU Parallel Loop Replace Examples

Bash Loop

```
for i in {1..10}; do  
    echo $i;  
done
```

```
for i in {1..100}; do  
    echo $i | grep 1$;  
done
```

GNU parallel Replacement

```
seq 1 10 | parallel echo {}
```

```
seq 1 100 | parallel 'echo {} | grep 1$'
```

Your Turn

Setup:

- On Janus, load GNU parallel
 - `module load gnu_parallel`
- Accept citation agreement
 - `parallel --citation`
 - will cite

Your Turn

Make this loop run in parallel with GNU parallel:

```
for i in {1..10}; do
    python print_input.py file_$(i).csv;
done
```

Hint: use {} instead of \$i:
file_{}.csv

```
print_input.py
```

```
import sys
print(sys.argv)
```

Previous Example

```
for i in {1..10}; do
    echo $i;
done
```

```
seq 1 10 | parallel echo {}
```

Solution

Original	<pre>for i in {1..10}; do python print_input.py file_\$.csv; done</pre>
GNU Parallel	<pre>seq 1 10 parallel python print_input.py file_{}.csv</pre>
Output	<pre>['print_input.py', 'file_1.csv'] ['print_input.py', 'file_2.csv'] ... ['print_input.py', 'file_10.csv']</pre>

GNU Parallel Useful Options

View what commands parallel will run without executing them:

```
$ seq 10 | parallel --dry-run echo {}
```

Limit number of tasks running at one time:

```
$ seq 10 | parallel -j 2 echo {}
```

Wait until enough memory is available to start next task:

```
$ seq 10 | parallel --memfree 2G echo {}
```

See all the options:

```
$ man parallel
```

GNU Parallel with Slurm

submit_gnu_parallel.sh

```
#!/bin/bash
#SBATCH --partition janus
#SBATCH --job-name gnu_parallel
#SBATCH --nodes 1
#SBATCH --output gnu_parallel.out
#SBATCH --reservation scbasics2
#SBATCH --time 01:00:00

# The following should be on one line
seq 10 | parallel python print_input.py file_
{}.csv
```

print_input.py

```
import sys
print(sys.argv) # print command line input

# process data here
```

GNU Parallel Summary

- Great for replacing loops and speeding them up
 - Control how your tasks are run
 - Can run on multiple computers as well (may take some effort to get working with slurm)
 - Lots of examples and documentation online
 - Useful tool outside of compute nodes too
-
- Takes time to learn
 - You may have to install a local copy on other systems.

Load Balancer

- Submitting hundreds of slurm jobs is inefficient
- RC provides a utility that balances serial applications using MPI (without needing knowledge of MPI!).
- The loadbalancer schedules tasks across multiple nodes after submitting one job
 - Choose how many tasks will run at a time
 - Starts tasks in order (no control over output order)
 - Replaces finished tasks with new ones
 - Straightforward input format

Load Balancer Create Input File

- Create a file with one task per line
 - Each task may be composed of multiple commands, each command separated by a semicolon

```
for i in {1..100}; do  
    echo "sleep 2; echo process $i" >> cmd_file ;  
done
```

Load Balancer Input File Example

cmd_file

sleep 2; echo process 1

sleep 2; echo process 2

sleep 2; echo process 3

...

sleep 2; echo process 98

sleep 2; echo process 99

sleep 2; echo process 100

Submitting Jobs with lb

Using srun

```
$ module load slurm  
$ module load loadbalance  
$ srun -N 2 --ntasks-per-node=12 lb cmd_file
```

Using batch script

```
$ module load slurm  
$ module load loadbalance  
$ nano submit_lb.sh # Create sbatch script  
#!/bin/bash  
#SBATCH --nodes 2  
#SBATCH --ntasks-per-node 12  
#SBATCH --output output.out  
#SBATCH --partition janus  
  
srun lb cmd_file  
  
$ sbatch submit_lb.sh
```

Your Turn

Write an input file for the load balancer.

- Input file should be called `cmd_file`
- The input file should have at least 2 commands per line
 - One command should be `'hostname'`
 - One command should be `'sleep 2'`
- Example line:
 - `hostname; python print_input.py file_1.csv; echo 1; sleep 2;`
- Should be 50 lines long
- No copy/paste coding! Use a loop.
- Hint: `'echo'` and `'>>'` are useful commands
- Hint: `""` tells bash it's a string and not a command

Two Possible Solutions

```
for i in {1..50}; do  
    echo "hostname; sleep 2; echo process $i; python print_input.py $i" >> cmd_file ;  
done
```

The following should be on one line

```
seq 1 50 | parallel 'echo "hostname; sleep 2; echo process {}; python print_input.py {};" >>  
cmd_file
```

Your Turn

Write a batch script and submit a job using the loadbalancer with `cmd_file` as the input file.

- Limit your job runtime to 2 minutes
- `scbasics2` reservation
- Output file `loadbalance.out`
- 1 node
- 12 tasks per node
- Call your batch script `submit_lb.sh`
- Hint: You need 'slurm' and 'loadbalance' modules loaded
- Hint: '`srun lb cmd_file`' will be used in your batch script

Possible Solution

```
submit_lb.sh
```

```
#!/bin/bash  
#SBATCH --nodes 1  
#SBATCH --ntasks-per-node 12  
#SBATCH --output loadbalance.out  
#SBATCH --time 00:02:00  
#SBATCH --reservation scbasics2
```

```
srun lb cmd_file
```

Submit with:
sbatch submit_lb.sh

Output from Multiple Nodes

Ran on 5 nodes with 5 tasks per node.

Input file with 'sleep 2; echo process \$i'

process 1

process 2

process 4

process 3

process 5

process 6

process 8

process 7

process 9

Load Balancer Summary

- No mpi knowledge required
- Saves time by reducing slurm overhead (and queue times for everyone)
- Runs on multiple nodes
- Input file can be created in your favorite language
- Not on other systems (it is on github)

Summary

- 1) Save yourself some time waiting in the queue by specifying a wall time on your jobs (--time)
- 2) Efficiently use resources to speed computation and allow more users to use the supercomputer at once. Run as many tasks as you can per node!

RC Resources:

- a) Bash scripting
- b) GNU parallel
- c) Load balancer

Questions?

Bash Script: <https://www.rc.colorado.edu/blog/reducejanuswaittimes>

GNU Parallel

Tutorial: https://www.gnu.org/software/parallel/parallel_tutorial.html

Examples: <https://www.gnu.org/software/parallel/man.html>

O. Tange (2011): GNU Parallel - The Command-Line Power Tool,
;login: The USENIX Magazine, February 2011:42-47.

Load Balancer: <https://www.rc.colorado.edu/support/examples-and-tutorials/load-balancer.html>

Additional Problems

Use GNU Parallel to parallelize the following loops:

Problem 1	Problem 2
<pre>for color in red green blue ; do for size in S M L XL XXL ; do echo \$color \$size done done</pre>	<pre>(for color in red green blue ; do for size in S M L XL XXL ; do echo \$color \$size done done) sort</pre>

Solutions

Problem 1	<code>parallel echo {1} {2} ::: red green blue ::: S M L XL XXL</code>
Problem 2	<code>parallel echo {1} {2} ::: red green blue ::: S M L XL XXL sort</code>

GNU Parallel vs Loop

```
$ seq 1 3 | parallel 'echo {}'; echo  
$$'
```

```
1          #loop number
```

```
27662      #process id
```

```
2
```

```
27663
```

```
3
```

```
27664
```

```
$ for i in {1..3}; do echo $i; echo  
$$; done
```

```
1
```

```
20614
```

```
2
```

```
20614
```

```
3
```

```
20614
```

