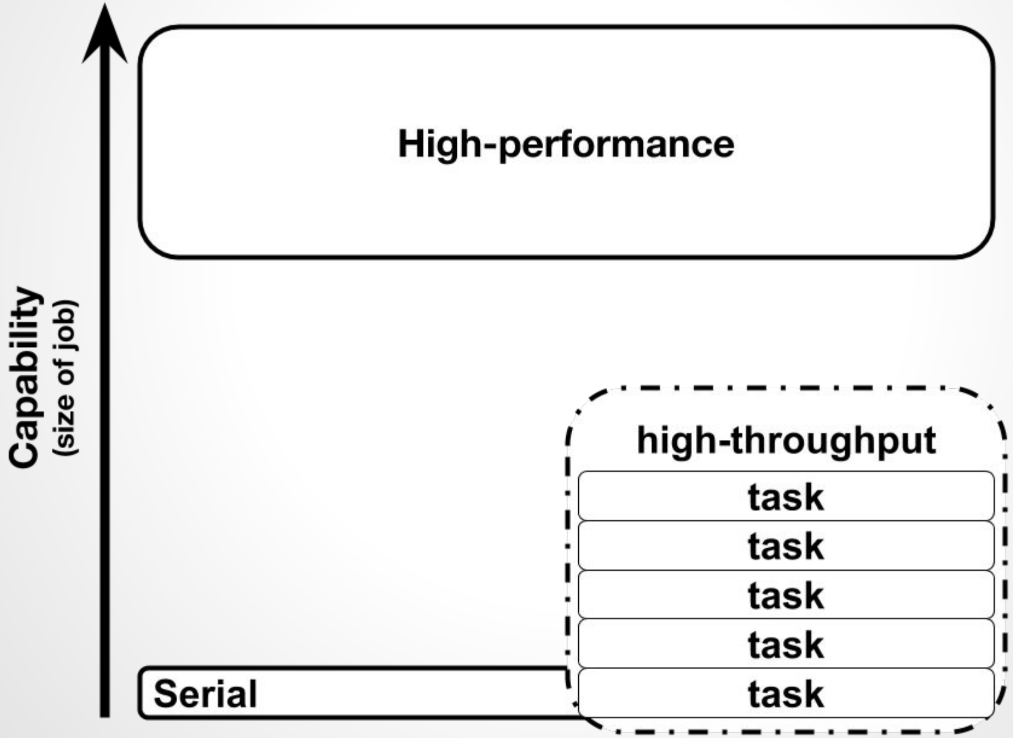
Efficient Submission of Serial Jobs

Aaron Holt

Slides and Examples: https://github.com/AaronTHolt/HTC_RMACC_17

HPC vs HTC


High-performance
Capability
(size of job)
high-throughput
task
task
task
task
task
Serial

Tools

slurm
workload manager

BASH
THE BOURNE-AGAIN SHELL

CURC loadbalancer

Workflow

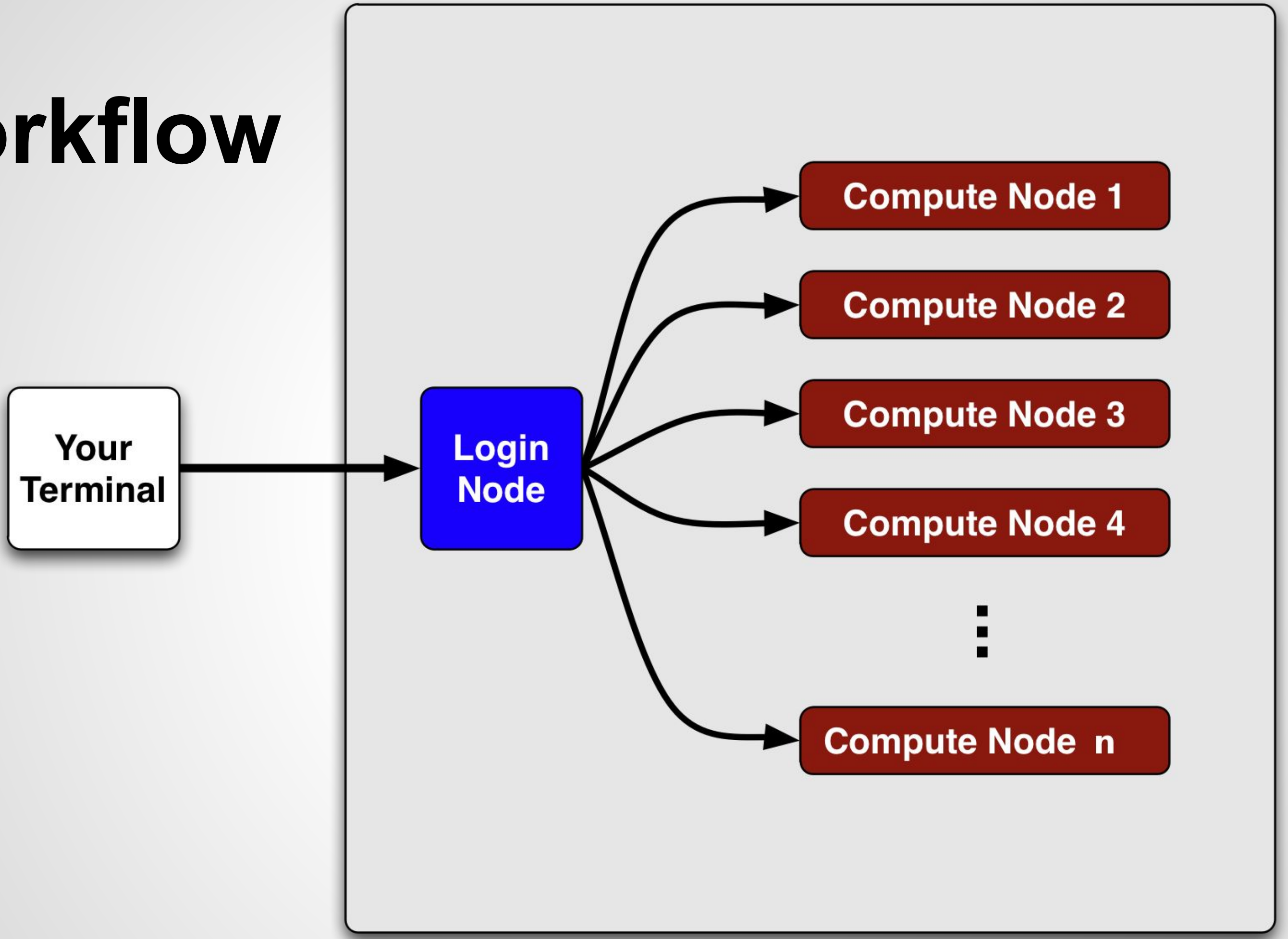


Image Source: https://docs.rice.edu/confluence/display/CD/Getting+Started+on+Blue+BioU

Setup

- Login to Sandstone HPC (Chrome and Firefox supported)
  - https://sandstone.rc.colorado.edu/
  - Start My Server
  - Start

- Those not using Sandstone can login with an ssh client
  - ssh USERNAME@login.rc.colorado.edu

Setup 2

- In the terminal app
  - git clone https://github.com/AaronTHolt/HTC_RMACC_17
  - ‘cd’ into cloned directory
  - ‘module load slurm/summit’

Batch Script with One Serial Task

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --partition sknl
#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
#!/bin/bash #SBATCH --partition sknl #SBATCH --nodes 1 #SBATCH --ntasks-per-node 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	#!/bin/bash #SBATCH --partition sknl #SBATCH --nodes 1 #SBATCH --ntasks-per-node **12** **#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**

Your Turn

In the Sandstone editor tab, edit the ‘background_exercise.sh’ file (located in HTC_RMACC_17/bash). Edit the file to accomplish the following:

1. Commands should run in parallel (use ‘&’ to background)
2. 2 minute wall time
3. Use 1 node in the sknl partition
4. Modify --ntasks-per-node so that each task has a cpu
5. After editing: In the terminal, navigate to the ‘bash’ directory inside your cloned git directory (‘cd bash’). Submit your job with ‘sbatch background_exercise.sh’

Solution

background_exercise.sh

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 6
#SBATCH --time 0:02:00
#SBATCH --partition sknl
#SBATCH --reservation htc-tutorial
./matrix_mul &
./matrix_mul &
./matrix_mul &
./matrix_mul &
./matrix_mul &
./matrix_mul &
wait
```

Submit with:

```
sbatch background_exercise.sh
```

Bash Script Summary

- You don’t need a special tool
- Available almost everywhere

- Takes some experience to write more complex scripts
- Not great for running large numbers of tasks

Slurm Job Arrays

- Submit multiple sub-jobs from a single job script
- Array indices specified by --array
  - For example, #SBATCH --array 0-9,12,15
- Array index is available as $SLURM_ARRAY_TASK_ID (in the job)
- Master job id is available as $SLURM_ARRAY_JOB_ID (in the job)

Slurm Job Array ‘Hello World’

```
#!/bin/bash
#SBATCH --nodes 1
#SBATCH --partition sknl
#SBATCH --array 0-2
#SBATCH --output slurm-array-%A.%a.out
#SBATCH --reservation htc-tutorial

echo "Master job id: ${SLURM_ARRAY_JOB_ID}"
echo "Array index: ${SLURM_ARRAY_TASK_ID}"
```

Submit a Slurm Job Array

- In the terminal app, navigate to 'HTC_RMACC_17/slurm_arrays'
- Submit a job with sbatch in the terminal:
  - sbatch simple_job_array.sh

Exercise - Slurm Job Array

There are 10 files called file_{1..10}.txt located in 'HTC_RMACC_17/slurm_arrays/data'. Edit array_exercise.sh (in slurm_arrays directory) so that the print_file_contents.py python script processes each file. For example, the first job index should run the following:

- python print_file_contents.py data/file_1.txt

Hints:

- Replace the file number with ${SLURM_ARRAY_TASK_ID}
- Submit the job with 'sbatch array_exercise.sh'
- Specify a range of indices with a hyphen (ex: #SBATCH --array 0-9)

Slurm Job Array Summary

- Run on multiple nodes
- No additional tools/software required
- Similar features exist on other schedulers

- Incurs additional scheduling overhead (inefficient for many small tasks)

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 Example

```
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

submit_lb.sh	Submitting from the terminal

`#!/bin/bash`
`#SBATCH --nodes 3`
`#SBATCH --ntasks-per-node 6`
`#SBATCH --time 00:05:00`
`#SBATCH --output lb-output.out`
`#SBATCH --partition sknl`
`#SBATCH --reservation htc-tutorial`
 	
`module load loadbalance`
`mpirun lb cmd_file`
	`$ sbatch submit_lb.sh`

Your Turn

Write an input file for the load balancer.

- Input file should be called cmd_file_2
- 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; 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

Possible Solution

```
for i in {1..50}; do
	echo “hostname; sleep 2; echo process $i” >> cmd_file_2 ;
done
```

Your Turn

Edit the ‘exercise_submit_lb.sh’ batch script and submit a job using the loadbalancer with cmd_file_2 as the input file.

- Limit your job runtime to 2 minutes
- Output file loadbalance.out
- 2 nodes
- 3 tasks per node
- Hint: You need to load the ‘loadbalance’ module
- Hint: ‘mpirun lb FILENAME’ will runs FILENAME with the loadbalancer

Possible Solution

exercise_submit_lb.sh

```
#!/bin/bash
#SBATCH --nodes 2
#SBATCH --ntasks-per-node 3
#SBATCH --output loadbalance.out
#SBATCH --time 00:02:00
#SBATCH --partition sknl
#SBATCH --reservation htc-tutorial

module load loadbalance/0.2
mpirun lb cmd_file_2
```

Submit with:

```
sbatch exercise_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

- Save yourself some time waiting in the queue by specifying a wall time on your jobs (--time)
- Use the following resources to efficiently submit many serial jobs at once:
  a) Bash
  b) Slurm
  c) CURC Load Balancer

Questions?

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

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

Slurm: https://slurm.schedmd.com/

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.

GNU Parallel

- 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!
- Multi-node isn’t as good as the Load Balancer.

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 Summit, 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	for i in {1..10}; do
	    python print_input.py file_$i.csv;
	done

GNU Parallel	seq 1 10 | parallel python print_input.py file_{}.csv
Output	['print_input.py', 'file_1.csv'] ['print_input.py', 'file_2.csv'] … ['print_input.py', 'file_10.csv']

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	print_input.py
#!/bin/bash #SBATCH --job-name gnu_parallel #SBATCH --nodes 1 #SBATCH --output gnu_parallel.out #SBATCH --time 01:00:00 # The following should be on one line seq 10 | parallel python print_input.py file_{}.csv	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.

Additional Problems

Use GNU Parallel to parallelize the following loops:

Problem 1	**Problem 2**

`for color in red green blue ; do`
`    for size in S M L XL XXL ; do`
`        echo $color $size`
`    done`
`done`
	`(for color in red green blue ; do`
	`    for size in S M L XL XXL ; do`
	`        echo $color $size`
	`    done`
	`done) | sort`

Solutions

Problem 1	parallel echo {1} {2} ::: red green blue ::: S M L XL XXL
Problem 2	parallel echo {1} {2} ::: red green blue ::: S M L XL XXL | sort

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
```