Using Slurm in Parallel Computing

Peter Ruprecht
peter.ruprecht@colorado.edu

www.rc.colorado.edu

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

Outline

- Preliminaries – getting logged in and downloading repo
- Why is job scheduling needed?
- Overview of Slurm commands and directives
- Submitting our first job!
- Summit partitions and QoS
- OpenMP parallel job
- MPI parallel job
- Job arrays
- Creating pipelines with job dependencies

Preliminaries – logging in

Username/password on printed strips

Username is `user00XY`

Login node is `tutorial-login.rc.colorado.edu`

```
ssh user00XY@tutorial-login.rc.colorado.edu
```

```
git clone
https://github.com/ResearchComputing/Parallel
ization_Workshop
```

What is a compute job?

- A unit of computing work handled by the scheduler
- Normally an execution of a program with a single input or single set of parameters; may have “steps”
- Interactive job
  - Allows you to work interactively at the command line of a compute node (or nodes)
  - Request needs to be submitted to the scheduler
- Batch job
  - Job that is executed in the background without further user input
  - Create a text file containing information about the job’s resource requirements and what program should be run

Why schedule jobs?

- On a shared system, like Summit, jobs are scheduled rather than just run instantly at the command line
  - Jobs wait in a queue until resources are available
  - Jobs from different users don’t overlap each other
- Prioritize certain jobs if needed
- Account for system usage
- Spreads out the workload throughout the day and week

Slurm

- Resource manager
  - Keeps track of what compute nodes are available and how busy each is
  - Allocates job access to nodes as prompted by scheduler
- Scheduler
  - Manages a queue of pending jobs
  - Determines job priority and schedules jobs for running when resources are available
- Tools
  - Provides framework for submitting and monitoring jobs

Slurm commands

- `sbatch` – submits a batch file to the queue
- `salloc` – requests an allocation of compute nodes
- `squeue` – checks the status of the queue
- `sacct` – queries the Slurm accounting database
  - Historical information about jobs, including
  - start/end times
  - memory/CPU used
- `scancel` – cancels a queued or running job
- `sprio` – lists priority of pending jobs
- `srun` – launch a task as a job step

- `sinteractive` – requests an interactive job

Slurm flags or directives

- Flag – command line argument
- Directive – used in a batch script with `#SBATCH`
- `--nodes` – total number of nodes required
- `--time` – “wall time”
- `--ntasks` – total number of simultaneous tasks
  - Think of this as MPI ranks, or CPU cores requested
- `--ntasks-per-node` – this times `--nodes` equals `--ntasks`
- `--mem` – memory needed per node
- `--output` – file that contains stdout and stderr from the job
- `--reservation` – send the job to reserved nodes

More Slurm flags

- `--partition` – what group of nodes to run on
- `--qos` – "Quality of Service"
- `--mail-type` – at what point to email job info
  - Could be `BEGIN, END, FAIL, ALL`
- `--mail-user` – address to email job info to
- `--job-name` – text identifier for the job
- `--account` – what allocation account to charge

- `%j` – expands to the Job ID ; useful for differentiating output files

Submit an interactive job

- `ssh scompile`
- `sinteractive --reservation=parallelD1`
- When shell prompt appears, type:

```
hostname
ls
w
squeue –u $USER
```

- Since we didn't specify how many nodes or cores we needed, receive the default:
  - One core on one node
  - Four hour wall time
  - "shas" partition – general compute nodes ("Haswell"), which have 24 cores per node

Modify the interactive job

- `sinteractive --reservation=parallelD1 --time=00:03:00`

- When shell prompt appears, type:

```
env | grep –i slurm
squeue –u $USER
scontrol show job JOBID
```

Whole-node interactive job

- `sinteractive --reservation=parallelD1 --ntasks=24`

Summit Partitions

Partition	Description	# of nodes	cores/node	GPUs/node
shas	General Compute (Haswell)	380	24	0
sgpu	GPU-enabled nodes	10	24	effectively 4
smem	High-memory nodes	5	48	0
sknl	Phi (Knights Landing) nodes - [not currently available]	20	68	0

Summit Quality of Service

QoS	Description	Maxwall	Max jobs/user	Max nodes/user
normal	Default QoS	Derived from partition	n/a	256
debug	For quick turnaround when testing	1 H	1	32
long	For jobs needing longer wall times	7 D	n/a	20
condo	For groups who have purchased Summit nodes	7 D	n/a	n/a

Batch job example

```
#!/bin/bash
#SBATCH --nodes=1                     # Number of requested nodes
#SBATCH --time=0:05:00                # Max wall time
#SBATCH --qos=debug                   # Specify debug QOS
#SBATCH --partition=shas              # Specify Summit haswell nodes
#SBATCH --ntasks=24                   # Number of tasks per job
#SBATCH --job-name=Matlab_Gen_Parallel          # Job name
#SBATCH --output=MATLAB_GEN_PARALLEL.%j.out     # Output file name with Job ID

# Written by: Shelley Knuth
# Date: 24 February 2014

# purge all existing software modules
module purge

# load the matlab module
module load matlab/R2016b

# Change to the directory that the job should start in
cd /projects/$USER/tutorials/parallelization_workshop/new

# Run matlab without a GUI and ask for all available workers
matlab -nosplash -nodesktop -r "clear; num_workers=$SLURM_NTASKS; parallel_std;"
```

Exercise 1 – create and submit simple batch job

- `cd Slurm`
- Job script should be named hostname.sh
- Request 1 node and 4 cores
- Wall time of 2 minutes
- Should execute the command “hostname”
- Send output to a file called “hostname.JOBID.out”
- Job name is “hostname”
- Specify the “parallelD1” reservation
- Use `sbatch hostname.sh` to submit the job

Exercise 1 - answer

```
#!/bin/bash
#SBATCH --nodes=1                 # Number of requested nodes
#SBATCH --ntasks=4                # Number of tasks per job; ie number of cores
#SBATCH --time=0:02:00                    # Max wall time
#SBATCH --reservation=parallelD1      # Specify WORKSHOP reservation
#SBATCH --partition=shas                  # Specify Summit haswell nodes
#SBATCH --job-name=hostname               # Job name
#SBATCH --output=hostname.%j.out      # Output file name with Job ID

# Written by: You!
# Date: 15 May 2017

hostname
```

Exercise 2 – OpenMP job

- `cp hostname.sh openmp.sh`
- Edit openmp.sh
- Request 1 node and 4 cores
- Wall time of 2 minutes
- Tell OpenMP to use 4 cores
- Should execute the command “openmp-hello.x”
- Send output to a file called “openmp.JOBID.out”
- Job name is “openmp”
- Specify the “parallelD1” reservation
- Use `sbatch openmp.sh` to submit the job

Exercise 2 - answer

```
#!/bin/bash
#SBATCH --nodes=1                  # Number of requested nodes
#SBATCH --ntasks=4                 # Number of tasks per job; ie number of cores
#SBATCH --time=0:02:00                     # Max wall time
#SBATCH --reservation=parallelD1           # Specify WORKSHOP reservation
#SBATCH --partition=shas                   # Specify Summit haswell nodes
#SBATCH --job-name=openmp                  # Job name
#SBATCH --output=openmp.%j.out             # Output file name with Job ID

cd /home/$USER/Parallelization_Workshop/Day1/Slurm
module purge
module load intel/16.0.3
export OMP_NUM_THREADS=4
./openmp-hello.x
```

Exercise 3 – email notification

- Edit openmp.sh
- Same as before, except modify it to email you when the job starts and finishes

```
--mail-type=begin,end
--mail-user=first.last@somewhere.edu
```

- Use `sbatch openmp.sh` to submit the job

Exercise 4 – MPI job

- `cp openmp.sh mpi.sh`
- Edit mpi.sh
- Request 1 node and 24 cores
- Wall time of 2 minutes
- Should execute the command “mpi-hello.x”
- Send output to a file called “mpi.JOBID.out”
- Job name is “mpi”
- Specify the “parallelD1” reservation
- Also need to load “impi” module
- Use `sbatch mpi.sh` to submit the job

Exercise 4 - answer

```
#!/bin/bash
#SBATCH --nodes=1                # Number of requested nodes
#SBATCH --ntasks=24              # Number of tasks per job; ie number of cores
#SBATCH --time=0:02:00                   # Max wall time
#SBATCH --reservation=parallelD1         # Specify WORKSHOP reservation
#SBATCH --partition=shas                 # Specify Summit haswell nodes
#SBATCH --job-name=mpi                   # Job name
#SBATCH --output=mpi.%j.out              # Output file name with Job ID

cd /home/$USER/Parallelization_Workshop/Day1/Slurm
module purge
module load intel/16.0.3
module load impi
./mpi-hello.x –np 24
```

Job arrays

- A collection of batch jobs with identical resource requirements
- Useful for running the same program against multiple input data files
- Easy way to submit multiple independent jobs
- Identified with main JobID plus TaskID (ie, index number)

- `--array=<index-range>`
- (can be, eg, `1-10` , `1,2,3,5,8,13` , `1-9:2`)
- `--output=myjob.%A_%a.out`
- `$SLURM_ARRAY_TASK_ID` variable

Job array example

```
#!/bin/bash
#SBATCH --nodes=1                   # Number of requested nodes
#SBATCH --ntasks=1                  # Number of tasks per job; ie number of cores
#SBATCH --time=0:22:00                      # Max wall time
#SBATCH —array=1-10                         # Specify array tasks
#SBATCH --reservation=parallelD1            # Specify reservation
#SBATCH --partition=shas                    # Specify Summit haswell nodes
#SBATCH --job-name=analyze_exp              # Job name
#SBATCH --output=analyze.%A_%a.out          # Output file name with Job/Task ID

cd /home/$USER/Parallelization_Workshop/Day1/Slurm
module purge
module load intel/16.0.3
echo “I am Task ID: ” $SLURM_ARRAY_TASK_ID
./analyze.py exp_run.$SLURM_ARRAY_TASK_ID  # One input file per Array Task
```

Job dependencies

- Allows you to build a sequential set of jobs, or “pipeline”
- Subsequent jobs won’t start until state of previous jobs meet certain conditions

- `sbatch --dependency=type:jobid[:jobid]`

- Types include:

`after:jobid[:jobid]`	(starts after jobid has started)
`afterok:jobid[:jobid]`	(starts after jobid finishes ok)
`singleton`	(starts after all previous jobs with the same name and user have ended)

Job dependency example

- Submit one job, note its JobID, then submit subsequent jobs with dependency on that JobID

```
$ sbatch job1.sh
87732
$ sbatch --dependency=afterok:87732 job2.sh
87733
$ sbatch --dependency=afterok:87733 job3.sh
87734
```

Job dependency – job script

```
#!/bin/bash
#SBATCH --nodes=1                # Number of requested nodes
#SBATCH --ntasks=1               # Number of tasks per job; ie number of cores
#SBATCH --time=0:02:00                   # Max wall time
#SBATCH --reservation=parallelD1         # Specify reservation
#SBATCH --partition=shas                 # Specify Summit haswell nodes
#SBATCH --job-name=job1                  # Job name
#SBATCH --output=job1.%j.out             # Output file name with Job ID

cd /home/$USER/Parallelization_Workshop/Day1/Slurm
echo “job1 starting” `date`
sleep 60
echo “job1 ending” `date`
```

Thank you!

- Email rc-help@colorado.edu
- Twitter: CUBoulderRC

- Link to survey on this topic:

http://tinyurl.com/curc-survey16

- Slides:
  https://github.com/ResearchComputing/Parallelization_Workshop