Duke Compute Cluster Workshop

12/10/2015
Tom Milledge
https://rc.duke.edu/
rescomputing@duke.edu
Outline of talk

• Overview of Research Computing resources
• Duke Compute Cluster overview
• Introduction to SLURM basics
• Running multi-core and parallel jobs
• Running jobs with dependencies
• Live demo and questions
Accessing the Duke Compute Cluster

• From on-campus (University or DUHS networks):
  ssh descr-slogin-01.oit.duke.edu
  (or descr-slogin-02)

• From off campus, first
  ssh login.oit.duke.edu
  and then ssh to an slogin node

• Alternatively, use the Duke VPN:
  https://oit.duke.edu/net-security/network/remote/vpn/
Copying files and directories

• Use scp or rsync for Linux or Mac workstations
• Use winscp for Windows: https://winscp.net
• Copying a file to the DCC (“push”)
  deptbox % scp data001.txt netid@dscr-slogin-01.oit.duke.edu:.
• Copying a file from the DCC (“pull”):
  deptbox % scp netid@dscr-slogin-01.oit.duke.edu:output.txt .
• Use either scp -r (small files) or rsync –av (large files)
• Pushing a directory:
  rsync –av dir1/ netid@dscr-slogin-01.oit.duke.edu: . or
  scp -r dir1/ netid@dscr-slogin-01.oit.duke.edu: .
• Pulling a directory:
  rsync –av netid@dscr-slogin-01.oit.duke.edu:~:/dir1 .
  scp -r netid@dscr-slogin-01.oit.duke.edu:~:/dir1 .
Duke Compute Cluster file systems

/dscrhome (a symlink to /hpchome/group)
• Primary storage on the Isilon X-series filers
• 250 GB group quota (typical)
• Two week tape backup (TSM)
/work
• Temporary storage for large data files
• Place for temporary files associated with running jobs (I/O)
• Not backed-up
• Subject to purges based on file age and/or utilization levels
• 100 TB total volume size, unpartitioned
/scratch
• File storage on the local node SATA drive
Slurm resources

• The DSCR wiki: “SLURM Queueing System”
  https://wiki.duke.edu/display/SCSC/SLURM+Queueing+System

• “Official” SLURM docs
  http://schedmd.com/slurmdocs

• Older SLURM documentation
  https://computing.llnl.gov/linux/slurm/slurm.html

• Comes up a lot in Google searches

• outdated – use schedmd.com instead
Running an interactive job

• Reserve a compute node by typing
  
  ```bash
  srun --pty bash -i
  ```

```
tm103@dscr-slogin-02 ~ $ srun --pty bash -i
srun: job 186535 queued and waiting for resources
srun: job 186535 has been allocated resources
```

```
tm103@dscr-encode-11 ~ $
```

```
tm103@dscr-encode-11 ~ $ squeue -u tm103
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
186535 common bash tm103 R 0:14 1 dscr-encode-11
```

• I now have an interactive session in the common partition on node dscr-encode-11
Slurm jobs run in “partitions”

- Most partitions are dept-owned machines
- These can only be used by members of the group
- Partitions can overlap
- Submit to partitions with “--partition=” or “-p”, e.g.
  ```bash
  #SBATCH --partition=dbchem (in a script) or
  srun -p econ --pty bash -i (interactively)
  ```
- The default partition is called “common”
- The common partition includes most nodes
- Submitting to a group partition gives “high-priority”
- A high-priority job can preempt (force-suspend) common partition jobs
SLURM commands

- **sbatch**
  Submit a batch job (like “qsub”)
- **#SBATCH**
  Specify job parameters (like “#$”)
- **squeue** (like “qstat”)
  Show lists of jobs
- **scancel** (like “qdel”)
  Delete one or more batch jobs
- **sinfo** (like “qhost”)
  Show info about machines
- **scontrol**
  Show cluster configuration information
sbatch

• Use "sbatch" (all lower case) to submit
  
sbatch test.q

• Use "#SBATCH" (upper case) in your scripts for scheduler directives, e.g.
  
  #SBATCH --mem=1000
  
  #SBATCH --output=matlab.out

• All SLURM directives can be given on the command line instead of the script.

• http://slurm.schedmd.com/sbatch.html
sbatch example

#!/bin/bash
#
#SBATCH --output=test.out
uname -n # print hostname

This prints the name of the compute node in the file "test.out"

tm103@dscr-slogin-02 ~/slurm $ sbatch simple.q
Submitted batch job 186554

tm103@dscr-slogin-02 ~/slurm $ cat test.out
dscr-compeb-14
Long-form commands example

#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=slurm.out
#SBATCH --mem=100 # 100 MB RAM
#SBATCH --partition=abyss # e.g.
uname -n # print hostname

• For a user in the “abyss” group, this job will run in high priority on an “dbchem” node.
Short-form commands example

- SLURM short commands don’t use “=“ signs

```
#!/bin/bash
#SBATCH -J test
#SBATCH --mem=1G
#SBATCH -p dbchem
```

```
# prints hostname
uname -n
```

```
#SBATCH -o slurm.out
#SBATCH --mem=1G  # 1GB RAM
#SBATCH -p dbchem  #only works for owners
```
Matlab example script

#!/bin/bash
#SBATCH –J matlab
#SBATCH –o slurm.out
#SBATCH --mem=4G # 4 GB RAM
/opt/apps/matlabR2015a/bin/matlab -nojvm -nodosplay -singleCompThread -r
my_matlab_program

• The ”–singleCompThread“ command is required to prevent uncontrolled multithreading
Slurm memory directives

- **This is a hard limit** – always request a little more
  \[ \text{--mem}=<MB> \]

- The amount of memory required per node
  \[ \text{--mem-per-cpu}=<MB> \]

- The amount of memory per CPU core
- For multi-threaded jobs
- Note: --mem and --mem-per-cpu are mutually exclusive
Slurm parallel directives

- All parallel directives have defaults of 1
- `-N <number>` How many nodes (machines)
- `-n <number>` or `--ntasks=<number>` How many parallel jobs (“tasks”)
- `-c, --cpus-per-task=<ncpus>`
- Use `-c` for multi-threaded jobs
- The `--ntasks` default is one CPU core per task, but the `--cpus-per-task` option will change this default.
Multi-threaded (multi-core) example

#!/bin/bash
#SBATCH –J test
#SBATCH –o test.out
#SBATCH –c 4
#SBATCH –mem-per-cpu=500  #(500 MB)
myApplication –n $SLURM_CPUS_PER_TASK

• The value of $SLURM_CPUS_PER_TASK is the number after “-c”
• This example starts a single, multi-threaded job that uses 4 CPU cores and 2 GB (4x500MB) of RAM
OpenMP multicore example

```bash
#!/bin/bash
#SBATCH --job-name=openmp-test
#SBATCH -o slurm.out
#SBATCH --cpus-per-task=4
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
myOpenMPapp  # will run on 4 CPU cores

• This sets $OMP_NUM_THREADS to the value of $SLURM_CPUS_PER_TASK
```
Slurm job arrays

- a mechanism for submitting and managing collections of similar jobs
- Job arrays are only supported for batch jobs and the array index values are specified using the --array or -a option
- [http://slurm.schedmd.com/job_array.html](http://slurm.schedmd.com/job_array.html)
Matlab job array example

#!/bin/bash
#SBATCH --output=matlab.out
#SBATCH --array=1-100
#SBATCH --mem=4G
/opt/apps/MATLAB/R2012b/bin/matlab
-nojvm -nondisplay -singleCompThread
-r "rank=${SLURM_ARRAY_TASK_ID};my_prog;quit"

• Start 100 Matlab programs, each with a different “rank”, e.g. 1,2, ... 100
Running MPI jobs

• Supported MPI versions
  - Intel MPI
  - OpenMPI
  - LAM MPI

• SLURM MPI jobs use “--ntasks=(num)“

https://wiki.duke.edu/display/SCSC/Running+MPI+Jobs
Compiling with OpenMPI

tm103@dscr-slogin-02 ~$
export PATH=/opt/apps/slurm/openmpi/bin:$PATH

which mpicc

/opt/apps/slurm/openmpi/bin/mpicc

mpicc -o openhello hello.c

ls -l openhello

-rw-r-xr-x 1 tm103 scsc 9184 Sep 2 12:26 openhello
# OpenMPI job script

#!/bin/bash
#SBATCH --ntasks=20
#SBATCH --output=openhello.out
export PATH=/opt/apps/slurm/openmpi/bin:$PATH
mpirun -n $SLURM_NTASKS openhello
OpenMPI example output

tm103@dscr-slogin-02 ~/misc/slurm/openmpi $ cat openhello.out
dscr-core-01, rank 0 out of 20 processors
dscr-core-01, rank 1 out of 20 processors
dscr-core-01, rank 2 out of 20 processors
dscr-core-01, rank 3 out of 20 processors
dscr-core-01, rank 4 out of 20 processors
dscr-core-03, rank 13 out of 20 processors
dscr-core-03, rank 14 out of 20 processors
dscr-core-03, rank 10 out of 20 processors
dscr-core-03, rank 11 out of 20 processors
dscr-core-03, rank 12 out of 20 processors
dscr-core-02, rank 8 out of 20 processors
dscr-core-02, rank 9 out of 20 processors
dscr-core-02, rank 5 out of 20 processors
...

#!/bin/bash

#SBATCH --ntasks=20
#SBATCH --output=intelhello.out

export I_MPI_PMI_LIBRARY=/opt/slurm/lib64/libpmi.so

source /opt/apps/intel/intel/intelvars.sh

srun -n $SLURM_NTASKS intelhello
Job dependencies

- [https://hcc-docs.unl.edu/display/HCCDOC/Job+Dependencies](https://hcc-docs.unl.edu/display/HCCDOC/Job+Dependencies)
- Start job “dep2” after job “dep1”
  
  ```
  $ sbatch dep1.q
  Submitted batch job 666898
  ```
- Make a note of the assigned job ID of dep1
  
  ```
  $ sbatch --dependency=afterok:666898 dep2.q
  ```
- Job dep2 will not start until dep1 finishes
Job dependencies with arrays

- Wait for specific job array elements
  
  ```bash
  sbatch --depend=after:123_4 my.job
  sbatch --depend=afterok:123_4:123_8 my.job2
  ```

- Wait for entire job array to complete
  
  ```bash
  sbatch --depend=afterany:123 my.job
  ```

- Wait for entire job array to complete successfully
  
  ```bash
  sbatch --depend=afterok:123 my.job
  ```

- Wait for entire job array to complete and at least one task fails
  
  ```bash
  sbatch --depend=afternotok:123 my.job
  ```
Live demo notes

- `df -h`
- `srun --pty bash -i`
- `squeue | more`
- `squeue -S S`
- `squeue -S S | grep -v PD`
- `squeue -u (NetID)`
- `sbatch (job script)`
- `scancel (job id)`
- `uname -n, sleep, top`
- `sinfo | grep -v common`
- `scontrol show node (node name)`
- `scontrol show job (job id)`