Outline of talk

• Overview of Research Computing resources
• Duke Compute Cluster overview
• Running interactive and batch jobs
• Using Slurm job arrays
• Running multi-core and parallel jobs
• Specifying job dependencies
• Live demo and questions
Duke Research Computing services & resources

• Cluster computing - Duke Compute Cluster
• Virtual machines - RAPID (Research Toolkits), “Clockworks”
• Data storage - Duke Data Commons (NIH),
• Research computing networks - high speed, Software-Defined Network, “Protected Network”
• Consultation
• Access to national resources (XSEDE, OSG)
• Education & training
• Technology development
The Duke Compute Cluster (DCC)

• Formerly known as the “DSCR”
• 667 compute nodes with approximately 14209 CPU cores (as of March 2018)
• Most nodes are purchased by labs and depts
• Some are provided by the University
• 500 TB of primary storage (Isilon X-series)
• Red Hat Enterprise Linux 7
• Uses the Slurm job queueing system
Accessing the Duke Compute Cluster

• From on-campus (University or DUHS networks):

\texttt{ssh NetID@dcc-slogin.oit.duke.edu}

  This will connect to one of the three DCC login nodes

• From off campus, first, 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”)
  ```bash
  scp data001.txt netid@dcc-slogin-02.oit.duke.edu:.
  ```
- Copying a file from the DCC (“pull”):
  ```bash
  scp netid@dcc-slogin-02.oit.duke.edu:output.txt .
  ```
- Use either scp -r (small files) or rsync –av (large files)
- Pushing a directory:
  ```bash
  rsync -av dir1/ netid@dcc-slogin-02.oit.duke.edu: .
  or
  scp -r dir1/ netid@dcc-slogin-02.oit.duke.edu: .
  ```
- Pulling a directory:
  ```bash
  rsync -av netid@dcc-slogin-02.oit.duke.edu:~/dir1 .
  scp -r netid@dcc-slogin-02.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
- 200 TB total volume size, unpartitioned

/datacommons
- Archival storage on the Isilon NL-series filers
- available for $82/TB/year
Slurm resources

• The DCC web pages
  https://rc.duke.edu/the-duke-compute-cluster/

“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
Slurm jobs run in “partitions”

• Most DCC partitions are dept-owned machines
• These can only be used by members of the group
• Submitting to a group partition gives “high-priority”
• Submit to partitions with “--partition=” or “-p“, e.g.

  #SBATCH –p (partition name) (in a script) or
  srun –p (partition name)--pty bash –i
  (interactively)

• The default DCC partition is called “common”
• The common partition gives “low-priority” to most ESX hosts
DCC Partitions

There are different DCC partitions to which batch jobs and interactive sessions can be directed:

- **common**, for jobs that will run on the DCC core nodes (up to 64 GB RAM).
- **gpu-common**, for jobs that will run on DCC GPU nodes.
- **Group partitions** (partition names varies), for jobs that will run on lab-owned nodes
- **scavenger**, for jobs that will run on lab-owned nodes in “low priority” (kill and requeue preemption).
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-core-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-core-11

- I now have an interactive session in the common partition on node dscr-encode-11
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=1G
  #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 --output=slurm.out
#SBATCH --error=slurm.err
#SBATCH --mem=100 # 100 MB RAM
#SBATCH --partition=abyss # e.g.
uname -n 1>&2 #prints hostname to the error file
• For a user in the “abyss” group, this job will run in high priority on an “abyss” lab node.
Short-form commands example

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

#!/bin/bash

#SBATCH -o slurm.out
#SBATCH -e slurm.err
#SBATCH --mem=4G  # 4 GBs RAM
#SBATCH -p somelab  #only for members

uname -n 1>&2  #prints hostname to the error file
Matlab example script

#!/bin/bash
#SBATCH –J matlab
#SBATCH –o slurm.out
#SBATCH --mem=4G  # 4 GB RAM
module load Matlab/R2018b
matlab -nojvm -nodisplay -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
  --mem=<MB>
- The amount of memory required per node
  --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

-\texttt{N} <\texttt{number}> How many nodes (machines)

-\texttt{n} <\texttt{number}> or \texttt{--ntasks}=<\texttt{number}> How many parallel jobs (“tasks”)

-\texttt{c}, \texttt{--cpus-per-task}=<\texttt{ncpus}>

• Use \texttt{-c} for multi-threaded jobs

• The \texttt{--ntasks} default is one CPU core per task, but the \texttt{--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

!/bin/bash
#SBATCH –J openmp-test
#SBATCH –o slurm.out
#SBATCH –c 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

• What are job arrays?
• --array (or -a) option
• SLURM_ARRAY_TASK_ID environment variable
• Example with sequentially named files
• Examples with non-sequentially named files
• “Unrolling” existing for loops
Slurm job arrays

- a mechanism for submitting and managing collections of similar jobs
- one job script and one application program
- Each can be up 100,000 job tasks on the DCC
- Job arrays are only supported for batch jobs
- Job array “tasks” must be independent

http://slurm.schedmd.com/job_array.html
--array (or –a)

For example, in a job script, add the line

```
#SBATCH --array=1-30
```

or, alternatively,

```
#SBATCH -a 1-30
```

to submit 30 job tasks. The job array indices can also be specified on the command line, e.g.

```
sbatch -a 1-30 myjob.sh
```
Array indices, cont.

The index values can be continuous, e.g.
\[-a\ 0-31\] (32 tasks, numbered from 0,1,2,...,31)
or discontinuous, e.g.
\[-a\ 3,5,7-9,12\] (6 tasks, numbers 3,5,7,8,9,12)
It can also be a single job task, e.g.
\[-a\ 7\]

The discontinuous notation is useful for resubmitting specific job tasks that had previously failed.
Each job task is assigned the environmental variable \$SLURM_ARRAY_TASK_ID set to its index value.

```
#!/bin/bash

echo $SLURM_ARRAY_TASK_ID
```

Submitted batch job 24845830

```
lut@dcc-slogin-02 $ sbatch -a 1-3 array-test.sh
```

```
lut@dcc-slogin-02 $ ls slurm-24845830*
```

```
lut@dcc-slogin-02 $ cat slurm-24845830*
```

```bash
1
```

```bash
2
```

```bash
3
```

```
lut@dcc-slogin-02 $ cat slurm-24845830*
```

```bash
1
```

```bash
2
```

```bash
3
```
Python job array example

#!/bin/bash
#SBATCH --e slurm.err
#SBATCH --array=1-5000
python myCode.py

$ cat test.py
import os
rank=int(os.environ['SLURM_ARRAY_TASK_ID'])
...

• Start 5000 Python jobs, each with a different “rank”, initialized from SLURM_ARRAY_TASK_ID
Matlab job array example

#!/bin/bash
#SBATCH -e slurm_%A_%a.err
#SBATCH -o slurm_%A_%a.out
#SBATCH -a 1-100
#SBATCH --mem=4G
module load Matlab/R2017b
matlab -nodisplay -singleCompThread
-r "rank=${SLURM_ARRAY_TASK_ID};my_prog;quit"

• Start 100 Matlab programs, each with a different “rank”, e.g. 1,2, ... 100
Processing continuously named files

- For the case with input (or output) file names of the form file1, file2, ..., fileN for \(-a\ 1-N\), e.g.

```bash
#!/bin/bash
#SBATCH -e slurm_%A_%a.err
#SBATCH -o slurm_%A_%a.out
mkdir out_\${SLURM_ARRAY_TASK_ID}
cd out_\${SLURM_ARRAY_TASK_ID}
myapp ../input_\${SLURM_ARRAY_TASK_ID}.txt
```

where output directories `out_1, out_2, ...` are created for input files `input_1.txt, input_2.txt, ...`
Processing discontinuously named files

• Process an existing file list, e.g. files.txt

```bash
#!/bin/bash
readarray -t FILES < files.txt
FILENAME=${FILES[(($SLURM_ARRAY_TASK_ID - 1))]}!
myapp $FILENAME
```

• Dynamically generate a file list from “ls”

```bash
#!/bin/bash
export FILES=($(ls -1 myfile*))
FILENAME=${FILES[(($SLURM_ARRAY_TASK_ID - 1))]}!
myapp $FILENAME
```
“Unrolling” for loops

• Original “serial” code (Python)
  fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21]
  for i in range(len(fibonacci)):
    print(i, fibonacci[i])

• Job array version
  import os
  i=int(os.environ['SLURM_ARRAY_TASK_ID'])
  fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21]
  #for i in range(len(fibonacci)):
  #  print(i, fibonacci[i])
  where the for loop is commented out and each job task is doing a single “iteration”
“Unrolling” for loop, cont.

```
#!/bin/bash

#SBATCH --e slurm.err
module load Python/2.7.11
python fibonacci.py
```

```
(1, 1)
(2, 1)
(3, 2)
(4, 3)
(5, 5)
(6, 8)
(7, 13)
(8, 21)
```
Running MPI jobs

• Supported MPI versions
  - Intel MPI
  - OpenMPI
• SLURM MPI jobs use “--ntasks=(num)“

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

tm103@dcc-slogin-02  ~ $ module load OpenMPI/2.1.0
OpenMPI 2.1.0

S103@dcc-slogin-02  ~ $ which mpicc

S103@dcc-slogin-03  ~ $ cd /dscrhome/tm103/misc/slurm/openmpi

S103@dcc-slogin-03  ~/misc/slurm/openmpi $ mpicc -o openhello hello.c

S103@dscr-slogin-02  ~/misc/slurm/openmpi ls -l
openhello

-rw-r-xr-x. 1 tm103 scsc 9184 Sep 1 16:08 openhello
OpenMPI job script

#!/bin/bash
#SBATCH -o openhello.out
#SBATCH -e slurm.err
#SBATCH -n 20
module load OpenMPI/2.1.0
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
GPU nodes

To run a GPU batch job, add the job script lines

```bash
#SBATCH -p gpu-common --gres=gpu:1
#SBATCH -c 6
```

To get an interactive GPU node session, type the command line

```bash
srun -p gpu-common --gres=gpu:1 -c 6 --pty bash -i
```

---

tm103@dscr-slogin-02 ~ $ srun -p gpu-common --gres=gpu:1 -c 6 --pty bash -i
tm103@dscr-gpu-01 ~ $ /usr/local/cuda-7.5/samples/1_Utilities/deviceQuery/
deviceQuery
...  
Detected 1 CUDA Capable device(s)

Device 0: "Tesla K80"
  CUDA Driver Version / Runtime Version: 7.5 / 7.5
  CUDA Capability Major/Minor version number: 3.7
  Total amount of global memory: 11520 MBytes (12079136768 bytes)
  (13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores
...
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
  `sbatch --depend=after:123_4 my.job`
  `sbatch --depend=afterok:123_4:123_8 my.job2`

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

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

• Wait for entire job array to complete and at least one task fails
  `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)`