HPCC New User Training
Getting Started on HPCC Resources
(Part 2/2)

Misha Ahmadian
High Performance Computing Center

Summer 2022
Outline

Part 2:

- Resource Allocation and Job Submission with SLURM
- Software builds and installation
- HPCC Policies
- Getting Help
Resource Allocation and Job Submission with
Introduction to Slurm

- Simple Linux Utility for Resource Management (SLURM):
  - Primary HPCC scheduler used in RedRaider
  - **Main entities:**
    1. **Nodes**: Physical computing resources
    2. **Partition**: A logical set of nodes
    3. **Jobs**: Allocations of resources assigned to a user for a specified amount of time
    4. **Job Steps**: sets of (possibly parallel) tasks within a job
    5. **Tasks**: Implies the requested/allocated computing resources to process(es) per job or job step
       (By default, each task refers to 1 CPU core)
Useful Slurm Commands:

- **sinfo:**
  - View information about nodes and partitions.

- **squeue:**
  - View information about jobs located in partitions.
  - Useful options to filter the output:
    - `-u <user>, --user=<user>`: Shows the list of jobs or job steps that belong to a specific user
    - `--me`: Shows the list of jobs or job steps that belongs to you (owner)
    - `-p <partition>, --partition=<partition>`: Filters the jobs within a partition.

- **srun:**
  - submits a job for execution or initiates job steps in real time.
  - **srun** has the same options as **sbatch** with a few more. (Please see the man page)
  - **srun** works similar to the “mpirun” and it can be replaced with “mpirun” as well.
Job Submission in Slurm

- **sbatch**: 
  - submits a job script for later execution.
    - The submitted job stays in the queue until the requested resources become available.
    - The job submission script is a text file that contains “#SBATCH” hints with `sbatch` command line options.

```bash
#!/bin/bash
#SBATCH –J MPI_test
#SBATCH –N 2
#SBATCH –ntasks-per-node=128
#SBATCH –o %x.%j.out
#SBATCH –e %x.%j.err
#SBATCH –p nocona

module load gcc/10.1.0 openmpi/3.1.6
mpirun ./my_mpi
```
## Job Submission in Slurm

### Job Submission Script Layout:

<table>
<thead>
<tr>
<th>Description</th>
<th>SLURM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the name for job</td>
<td>-J, --job-name=&lt;jobname&gt;</td>
</tr>
<tr>
<td>The name of the standard output file</td>
<td>-o, --output=&lt;filename pattern&gt;</td>
</tr>
<tr>
<td>The name of the standard error file</td>
<td>-e, --error=&lt;filename pattern&gt;</td>
</tr>
<tr>
<td>Define the queue (partition) name</td>
<td>-p, --partition=&lt;partition_names&gt;</td>
</tr>
<tr>
<td>Type of parallel env for job/task allocation</td>
<td>-N, --nodes=&lt;# of nodes&gt;</td>
</tr>
<tr>
<td></td>
<td>--ntasks-per-node=&lt;ntasks&gt;</td>
</tr>
<tr>
<td>Reserve memory per slot</td>
<td>--mem-per-cpu=&lt;size[K</td>
</tr>
<tr>
<td>Set the maximum job run time</td>
<td>-t, --time=<a href="">HH:MM:SS</a></td>
</tr>
<tr>
<td>Specify the cluster policy for this job</td>
<td>-A, --account=&lt;account&gt;</td>
</tr>
</tbody>
</table>
Job Submission in Slurm

- Select a partition:
  - Partition in Slurm groups physical nodes into a logical set and allows jobs to request for nodes’ resources from that partitions.
  - `-p, --partition=<partition_name>`

<table>
<thead>
<tr>
<th>Name</th>
<th># of Nodes</th>
<th>Type</th>
<th>Nodes</th>
<th>#Core/Node</th>
<th>#Mem/Node</th>
<th>Mem/Core</th>
<th>#GPU/node</th>
</tr>
</thead>
<tbody>
<tr>
<td>nocona</td>
<td>240</td>
<td>AMD ROME CPU</td>
<td>cpu-[23-26]-[1-60]</td>
<td>128</td>
<td>503 GB</td>
<td>3.9 GB</td>
<td>N/A</td>
</tr>
<tr>
<td>matador</td>
<td>20</td>
<td>Intel/Nvidia V100 GPU</td>
<td>gpu-[20-21]-[1-10]</td>
<td>40</td>
<td>376 GB</td>
<td>9.4 GB</td>
<td>2</td>
</tr>
<tr>
<td>gpu-build</td>
<td>1</td>
<td>Intel/Nvidia V100 GPU</td>
<td>gpu-20-11</td>
<td>32</td>
<td>187 GB</td>
<td>5.9 GB</td>
<td>1</td>
</tr>
<tr>
<td>toreador</td>
<td>11</td>
<td>AMD/Nvidia A100</td>
<td>gpu-20-[12-15],gpu-21-[11-17]</td>
<td>16</td>
<td>188 GB</td>
<td>11.8 GB</td>
<td>3</td>
</tr>
<tr>
<td>quanah</td>
<td>467</td>
<td>Intel Xeon Broadwell</td>
<td>cpu-[1-10]-[*]</td>
<td>36</td>
<td>188 GB</td>
<td>5.3 GB</td>
<td>N/A</td>
</tr>
<tr>
<td>xlquanah</td>
<td>16</td>
<td>Intel Xeon Broadwell</td>
<td>cpu-19-[1-16]</td>
<td>36</td>
<td>251GB</td>
<td>6.9 GB</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Job Submission in Slurm

• Requesting CPU:
  • In Slurm, users must define the following options in their job submissions in order to request for CPU resources:
    1. Number of nodes: How many total nodes for the job?
       - \(-N, \ --\text{nodes}=<\text{number of nodes}>\)
    2. Number of tasks per node: (Recommended) (By default, each task consumes 1x CPU core)
       - \(--\text{ntasks\_per\_nodes}=<\text{number of task per node}>\)
       OR Number of total tasks: How many task across the nodes?
       - \(-n, \ --\text{ntasks}=<\text{number of tasks}>\)

• Requesting the right number of cores is key to optimizing throughput
Job Submission in Slurm

• Requesting Memory:
  • One can specify the size of the consumable Memory in two ways in Slurm:
    1. Memory per core (*Recommended*):
       - `--mem-per-cpu=<size[M|G]>`
    2. Memory per node:
       - `--mem=<size[M|G]>`

• If no memory size was specified, Slurm will assign the default memory per core to your job.

<table>
<thead>
<tr>
<th>System</th>
<th>Memory Per Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nocona</td>
<td>4027 MB (3.9 GB)</td>
</tr>
<tr>
<td>Matador</td>
<td>9639 MB (9.4 GB)</td>
</tr>
<tr>
<td>Quanah</td>
<td>5370 MB (5.3 GB)</td>
</tr>
</tbody>
</table>

• Make sure you won’t exceed the total memory per node:
  • `--mem-per-cpu=100G`
Job Submission in Slurm

- **Requesting Runtime limits:**
  - Recommended that you set the max runtime you expect a job will take.
    - `-t, --time=<time>`
    - `<time>` can be:
      - `minute`
      - `minute:seconds`
      - `hours:minutes:seconds`
      - `days-hours`
      - `days-hours:minutes`
      - `days-hours:minutes:seconds`
  
  - E.g., `--time=24:00:00`

- Please note that there is a 48-hour default time limit per job and exceeding this amount will end up with rejecting your job submission.
Job Submission in Slurm

- Requesting GPU:
  - GPUs are available by requesting any node in the matador partition.
  - Number of GPUs per node (Recommended):
    - `--gpus-per-node=[<type>:]<number>`
  - Total number of GPUs:
    - `-G, --gpus=<# of gpus>` (Currently Unavailable on RedRaider cluster)
  - There is only one type of GPU in RedRaider Cluster (v100) and is optional to be specified.
  - It is required to requesting at least one GPU per node when submitting a job to Matador.
  - Make sure you do not exceed more than 2 GPUs per node during the job submission.
    - `--partition=matador --nodes=2 -gpus-per-node=4` ✗
    - `--partition=matador --nodes=2 --gpu-per-node=2` ✓
Job Submission in Slurm

- Accounts on RedRaider Cluster:
  - Accounts, in Slurm, imposes a set of pre-defined resource limits and assigns the usage/fair-share policies to each job.
    - `-A, --account=<account>`

<table>
<thead>
<tr>
<th>Account</th>
<th>Default Runtime</th>
<th>Maximum Runtime</th>
<th>CPU/Mem Limit per job</th>
<th>Total # Jobs Per User</th>
<th># Running Jobs / User</th>
<th>Allowed Partitions</th>
<th>Priority</th>
<th>Special Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>default *</td>
<td>48 hours</td>
<td>48 hours</td>
<td>No limit</td>
<td>2000</td>
<td>No limit</td>
<td>All Partitions</td>
<td>normal</td>
<td>No</td>
</tr>
<tr>
<td>xlquanah **</td>
<td>72 hours</td>
<td>14 days</td>
<td>36 cores / 251GB</td>
<td>100</td>
<td>3</td>
<td>xlquanah</td>
<td>normal</td>
<td>Yes</td>
</tr>
<tr>
<td>Dedicated resource users</td>
<td>72 hours</td>
<td>No limit</td>
<td>Up to the total available resources</td>
<td>No limit</td>
<td>No limit</td>
<td>Nocona/Quanah</td>
<td>high</td>
<td>Yes</td>
</tr>
</tbody>
</table>

(*) The system will assign the default Account/QoS if the user does not define them in their job submissions.
(**) Request for special access is required. There should be a valid use case to approve the access to the “xlquanah” partition.
Submitting Jobs

Example of a simple job to submit an MPI program to Slurm:

- Create a job submission script file (e.g., submit.sh):

```bash
#!/bin/bash
#SBATCH -J MPI_test
#SBATCH -N 2
#SBATCH -n tasks-per-node=128
#SBATCH -o %x.%j.out
#SBATCH -e %x.%j.err
#SBATCH -p nocona
module load gcc/10.1.0 openmpi/3.1.6
mpirun ./my_mpi
```

- Submit the job with `sbatch`:
  - `sbatch submit.sh`

- Monitor the job with `squeue`:
  - `squeue --me`
  - `squeue -u <username>`

- Cancel the job with `scancel`:
  - `scancel job_id`

Example: `/lustre/work/examples/nocona`
1. Make sure you’re already Logged in to the “login.hpcc.ttu.edu” using your eraider account.
2. Go to your home directory and copy the following directory into your home directory:
   
   ```
   $ cp -r /lustre/work/examples/nocona/training/mpi ~/
   ```

3. Go into the ‘mpi’ directory on your home directory:
   a) List the contents of the directory
   b) Print the contents of the ‘makefile’ file
   c) Load the proper modules for “GCC 10.1.0” and “OpenMPI 4.0.4”
   d) Run the “make” command to compile the “mpi_hello_world.c” code
   e) Modify the ‘mpi_slurm.sh’ file as follows:
      i. Request 1 node from ‘nocona’ partition with 2 tasks (CPU cores) per node
      ii. Load the right modules that will work properly with the “mpi_hello_world”
   f) Submit the ‘mpi_slurm.sh’ job script
   g) Check the current status of your jobs
   h) Check the job’s output/error files after it finished.
Interactive Session

- **interactive:**
  - Starts an interactive session/job:
    - `interactive -c 2 -p nocona`
    - See the `interactive -h` for all the available options.
  - Make sure the prompt changes to `cpu-#-#`.
  - Make sure you run “exit” when you’re finished.
  - Keep in mind resource/runtime limits apply to `interactive` based on the selected account.
  - The `interactive` command will forward the X11 if the SSH session was established with `-X` or `-Y`.
  - Please note that direct SSH to any worker nodes not part of your job is blocked on the RedRaider cluster.
Building and Testing GPU applications:

- The `gpu-build` partition contains one Intel/GPU node with **1x Nvidia V100 GPU device**, **32x Intel CPU cores** and **192 GB RAM**, which allows users to:
  - Build their own GPU applications.
  - Test GPU applications and the environment setup before submitting a job to Matador partition.
  - Accessing the Lmod Module environment for GPU compilers/applications.

- In order to access the `gpu-build` node, you need to establish an *interactive* session:
  - `$ interactive -p gpu-build -c 2`

Limitations:

<table>
<thead>
<tr>
<th>Partition</th>
<th>Max Runtime (per job)</th>
<th>Max CPU per user (in total)</th>
<th>Max Mem per user (in total)</th>
<th>Max interactive session per user</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpu-build</td>
<td>5 hours</td>
<td>6</td>
<td>36006 MB (35 GB)</td>
<td>2</td>
</tr>
</tbody>
</table>
Debugging a Finished Job

- **sacct:**
  - reports accounting information about active or completed jobs or job steps.
    - `sacct -j <jobid>`
  - More filter options are available by checking the `-e, --helpformat` options of `sacct` command.
    - `sacct -j <jobid> --format=partition,jobid,ntasks,nodelist,maxrss,maxvmsize,exitcode`
  - When debugging:
    - Check the output and error files
    - Check the output of `sacct` for:
      - Memory usage
      - Exit code
      - Start and end time.
Current Status of the Job Scheduler

- You can check the current status of the Slurm Job Scheduler at this [Link](#).
Exercise #3

1. Make sure you’re already Logged in to the “login.hpcc.ttu.edu” using your eraider account.
   - For this example, please assure your SSH session has the X11 forwarding enabled!
2. Make an interactive session to one of the Nocona nodes:
   - Use the ‘interactive’ command.
   - Request for 1 CPU core from ‘nocona’ partition.
   - You can use the same temporary reservation as you used in the last exercise.
3. Once the worker node was allocated, locate the “MATLAB” module
4. Try to run the MATLAB graphical user interface (GUI) on the cluster:
   - `cpu-#-#$ matlab`
5. Close the MATLAB window to exit the program.
6. Exit the interactive session.
Software builds and installation
• Multiple partitions – Multiple architectures:

<table>
<thead>
<tr>
<th>Nocona</th>
<th>Matador</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD EPYC ROME</td>
<td>Intel Xeon Cascade Lake</td>
</tr>
<tr>
<td></td>
<td>Nvidia V100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quanah</th>
<th>Toreador</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon Broadwell</td>
<td>AMD EPYC ROME</td>
</tr>
<tr>
<td></td>
<td>Nvidia A100</td>
</tr>
</tbody>
</table>
Software builds on HPCC Clusters

• Modules & compiled code are different on each of the RedRaider partitions!
  • Each CPU architecture may bring a different set of features and instructions.
  • Compiled programs (C/C++/Fortran) need to be re-compiled to match each CPU architecture.
  • E.g., programs that are compiled on Intel nodes may not work properly/efficiently on AMD nodes.
  • Different Compilers/Math libraries optimize the programs differently on each architecture.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>AMD ROME</th>
<th>Intel Broadwell</th>
<th>Intel Ivy Bridge</th>
<th>Intel Cascade Lake</th>
<th>Nvidia GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU/GCC</td>
<td>GCC 10+</td>
<td>GCC 4+</td>
<td>GCC 4+</td>
<td>GCC 10+</td>
<td>GCC 8+</td>
</tr>
<tr>
<td>Intel</td>
<td>May work</td>
<td>Optimized</td>
<td>Optimized</td>
<td>Optimized</td>
<td>Intel 19+</td>
</tr>
<tr>
<td>AOCC</td>
<td>Optimized</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
<td>N/A</td>
</tr>
<tr>
<td>MKL</td>
<td>May work</td>
<td>Optimized</td>
<td>Optimized</td>
<td>Optimized</td>
<td>MKL 19+</td>
</tr>
<tr>
<td>AOCL</td>
<td>Optimized</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
<td>Not Applicable</td>
<td>N/A</td>
</tr>
<tr>
<td>CUDA</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>CUDA 10+</td>
</tr>
</tbody>
</table>
Tips and Recommendations:

1. Create a separate directory for each CPU architecture, and make a copy from your code/program and place it under each directory:
   - mkdir nocona matador quanah

2. Login to the RedRaider login node, and for each CPU architecture make an interactive session to the corresponding worker node:
   - interactive -p nocona –c 10

3. Go to the directory of your code that has the same name as the current session’s partition:
   - cd nocona

4. Load a proper compiler module and recompile your code:
   - module load gcc/10.1.0

5. If applicable, add the -O3 optimization flag to all the CFLAGS, CPPFLAGS, CXXFLAGS, FFLAGS.
   - CFLAGS=-O3 FFLAGS=-O3 make -j 10 all
Software builds on HPCC Clusters

• Tips and Recommendations:

5. We recommend mapping the MPI jobs to the L3-cache memory on **Nocona (AMD)** nodes:
   - `mpirun -map-by l3cache ./mpi_app`

6. **HPCC will not support Python v2 on Nocona and Matador nodes with CentOS 8. (This rule will be applied to Quanah and Ivy in the near future.)**
   - Users can still get Python v2 from Conda (Anaconda/Miniconda)
   - Python 2 is NOT RECEIVING SECURITY UPDATES and should be retired from your workflows ASAP.

7. Python applications (including the applications from Condo repo) will continue working with different architectures without recompiling them.
Local Python Package Installation

- Install a Python package into your home folder:

  $ module load intel python
  $ pip install --user <package name>
  - Example: pip install --user matplotlib

- Install a local copy of Python using Conda:

  $ /lustre/work/examples/InstallPython.sh
  $ . $HOME/conda/etc/profile.d/conda.sh
  $ conda activate
  $ conda install <package name>
  - Example: conda install biopython
Local R Package Installation

• Install an R package into your home folder:

  ▪ Example (On Quanah Node:)
    $ module load intel R
    $ R
    $ install.packages(‘<package name>’)
    Example: install.packages(‘readr’)

  ▪ Select a mirror

  ▪ The R application will ask if you want to install it locally the first time you do this.
HPCC Policies
HPCC Policies

• **Login nodes** (login.hpcc.ttu.edu, quanah.hpcc.ttu.edu):
  • No jobs are allowed to run on the login node.

• **SSH Access:**
  • No direct SSH access allowed to a node(s) if you have no job running on the node(s)

• **Software Installation:**
  • Software requests are handled on a case-by-case basis
  • Requesting software does not guarantee it will be installed “cluster-wide”.
  • May take two or more weeks to complete your request.

• **Scratch Purge Policy:**
  • Scratch will be purged monthly by removing of all files not accessed within the past year, or sufficient to bring total scratch space usage across all accounts down to 80% of capacity.
Getting Help
Further Assistance

• Visit Our Website:
  • Most up-to-date user guide documents
    • https://www.depts.ttu.edu/hpcc/userguides/index.php
  • Job scheduler and resource allocation status page
    • https://www.depts.ttu.edu/hpcc/status/slurm_web.php
  • Current status of all HPCC services
    • https://www.depts.ttu.edu/hpcc/status/cachet.php

• Read the documentation!
  • https://slurm.schedmd.com/documentation.html

• Submit a support ticket:
  • Send an email to hpccsupport@ttu.edu
Quick Reminder

- **HPCC Training Courses**
  - Please check the website for upcoming User Training workshops
    - [http://www.depts.ttu.edu/hpcc/about/training.php](http://www.depts.ttu.edu/hpcc/about/training.php)

- **ShortCourse Survey**
  - Looking forward to have your feedback on this Training Workshop
    - You will receive a survey in your inbox from TTU ShortCourse

- **The PowerPoint slides are available online**
  - [http://www.depts.ttu.edu/hpcc/about/training.php](http://www.depts.ttu.edu/hpcc/about/training.php)