1 Introduction

SLURM (Simple Linux Utility for Resource Management) is a software package for submitting, scheduling, and monitoring jobs on large compute clusters. This page details how to use SLURM for submitting and monitoring jobs on ACCRE's Vampire cluster. New cluster users should consult our Getting Started pages, which is designed to walk you through the process of creating a job script, submitting a job to the cluster, monitoring jobs, checking job usage statistics, and understanding our cluster policies.

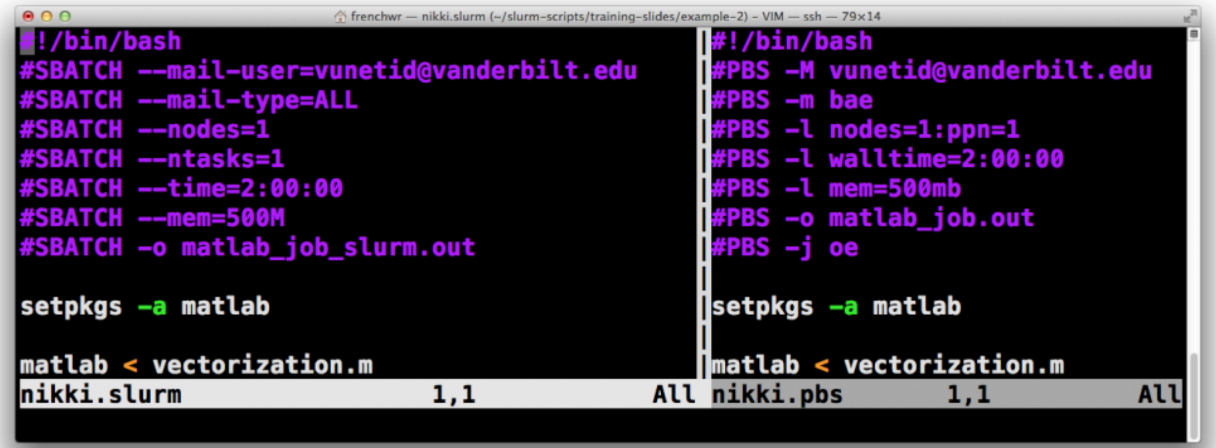
Until early 2015, Vampire used Torque for resource management and Moab for job scheduling, and users submitted a job to Vampire by writing a script specifying the resources and commands needed to execute a program. SLURM also requires users to submit jobs through a script, with slightly different syntax compared to Torque/Moab. These differences are highlighted in section 2 . A summary of SLURM commands is shown in section 3 . (A great reference for SLURM commands can also be found by clicking here .)

All the examples on this page can be downloaded from ACCRE's Github page by issuing the following commands from a cluster gateway:

```
setpkgs -a git
git clone https://github.com/accre/SLURM.git
```

2 SLURM vs. Torque

Converting a Torque batch script to SLURM is generally a straightforward process. Below is a simple SLURM script (lefthand side) for running a Matlab job requesting 1 node, 1 CPU core, 500 MB of RAM, and 2 hours of wall time. For comparison, the equivalent Torque script is shown on the right. Aside from syntax, the two scripts have only very minor differences. In general, `#SBATCH` options tend to be more self-explanatory. Note that specifying the node (`#SBATCH --nodes=1`) and CPU core (`#SBATCH --ntasks=1`) count must be broken off into two lines in SLURM, and that SLURM has no equivalent to `#PBS -j oe` (SLURM combines standard output and error into a single file by default).


frenchwr — nikki.slurm (~/slurm-scripts/training-slides/example-2) – VIM — ssh — 79×14

```
#!/bin/bash
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=2:00:00
#SBATCH --mem=500M
#SBATCH -o matlab_job_slurm.out

setpkgs -a matlab

matlab < vectorization.m
nikki.slurm                 1,1          All
```

```
#!/bin/bash
#PBS -M vunetid@vanderbilt.edu
#PBS -m bae
#PBS -l nodes=1:ppn=1
#PBS -l walltime=2:00:00
#PBS -l mem=500mb
#PBS -o matlab_job.out
#PBS -j oe

setpkgs -a matlab

matlab < vectorization.m
nikki.pbs        1,1          All
```

Screen Shot 2014-12-09 at 5.35.50 PM

Like Torque batch scripts, a SLURM batch script must begin with the `#!/bin/bash` directive on the first line. The subsequent lines begin with the SLURM directive `#SBATCH` followed by a resource request or other pertinent job information. Email alerts will be sent to the specified address when the job begins, aborts, and ends.

For reference, the following table lists common Torque options along side the equivalent option in SLURM. For examples of how to include the appropriate SLURM options for parallel jobs, please refer to Section 5.

Torque	SLURM	Meaning
`-l nodes=[count]`	`--nodes=[count]`	Node count
`-l ppn=[count]`	`--tasks-per-node=[count]`	Processes pe
	`--ntasks=[count]`	Total proces
	`--cpus-per-task=[count]`	CPU cores p
	`--nodelist=[nodes]`	Job host pr
	`--exclude=[nodes]`	Job host to
`-l walltime=[dd:hh:mm:ss]`	`--time=[min] or --time=[dd-hh:mm:ss]`	Wall clock l
`-l mem=[count]`	`--mem=[count]`	RAM per no
`-l pmem=[count]`	`--mem-per-cpu=[count][M or G]`	RAM per C
`-o [file_name]`	`--output=[file_name]`	Standard ou
`-e [file_name]`	`--error=[file_name]`	Standard er
`-j oe`	(default behavior)	Combine sto
`-t [array_spec]`	`--array=[array_spec]`	Launch job
`-M [email_address]`	`--mail-user=[email_address]`	Email for jo
`-m [a or b or e]`	`--mail-type=[BEGIN or END or FAIL or REQUEUE or ALL]`	Email alert
`-W group_list=[account]`	`--account=[account]`	Account to
`-d [job_id]`	`--depend=[state:job_id]`	Job depend
`-N [name]`	`--job-name=[name]`	Job name
	`--constrain=[attribute]`	Request noc
	`--partition=[name]`	Submit job

Note that the `--constrain` option allows a user to target certain processor families or nodes with a specific CPU core count. All non-GPU groups on the cluster have access to the `production` and `debug` partitions. The purpose of the `debug` partition is to allow users to quickly test a representative job before submitting a larger number of jobs to the production partition (which is the default partition on our cluster). Wall time limits and other policies for each of our partitions are shown below.

Partition	Max Wall Time	Max Running Jobs	Max Submitted Jobs	Resources
`production`	14 days	n/a	n/a	6000-6500 CPU cores
`debug`	30 minutes	2	5	8 CPU cores
`maxwell`	5 days	n/a	n/a	144 CPU cores, 48 Ma
`fermi`	14 days	n/a	n/a	128 CPU cores, 64 Fer
`mic`	14 days	2	n/a	64 CPU cores, 8 Intel

3 SLURM Commands

Just like Torque, SLURM offers a number of helpful commands for tasks ranging from job submission and monitoring to modifying resource requests for jobs that have already been submitted to the queue. Below is a list of SLURM commands, as well as the Torque equivalent in the far left column.

Torque	SLURM	Function
`qsub [job_script]`	`sbatch [job_script]`	Job submission
`qstat` or `showq`	`squeue`	Job/Queue status
`qdel [JOB_ID]`	`scancel [JOB_ID]`	Job deletion
`pbsnodes`	`scontrol show nodes`	Node list
`qhold [JOB_ID]`	`scontrol hold [JOB_ID]`	Job hold
`qrls [JOB_ID]`	`scontrol release [JOB_ID]`	Job release
`qstat -a`	`sinfo`	Cluster status
`qsub -I`	`salloc`	Launch interactive job
	`srun [command]`	Launch (parallel) job step
	`sacct`	Displays job accounting information

3.1 `sbatch`

The `sbatch` command is used for submitting jobs to the cluster. Like Torque's `qsub` , `sbatch` accepts a number of options either from the command line, or (more typically) from a batch script. An example of a SLURM batch script (called `simple.slurm`) is shown below:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=1G
#SBATCH --time=0-00:15:00     # 15 minutes
#SBATCH --output=my.stdout
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --job-name="just_a_test"

# Put commands for executing job below this line
# This example is loading Python 2.7.8 and then
# writing out the version of Python
setpkgs -a python2.7.8
python --version
```

To submit this batch script, a user would type:

```
sbatch simple.slurm
```

This job (called `just_a_test`) requests 1 compute node, 1 task (by default, SLURM will assign 1 CPU core per task), 1 GB of RAM per CPU core, and 15 minutes of wall time (the time required for the job to complete). Note that these are the defaults for any job, but it is good practice to include these lines in a SLURM script in case you need to request additional resources.

Optionally, any `#SBATCH` line may be replaced with an equivalent command-line option. For instance, the `#SBATCH --ntasks=1` line could be removed and a user could specify this option from the command line using:

```
sbatch --ntasks=1 simple.slurm
```

The commands needed to execute a program must be included beneath all `#SBATCH` commands. Lines beginning with the `#` symbol (without /bin/bash or SBATCH) are comment lines that are not executed by the shell. The example above simply prints the version of Python loaded in a user's path. It is good practice to include any `setpkgs` commands in your SLURM script. A real job would likely do something more complex than the example above, such as read in a Python file for processing by the Python interpreter.

For more information about `sbatch` see: http://slurm.schedmd.com/sbatch.html

3.2 `squeue`

`squeue` is used for viewing the status of jobs. By default, `squeue` will output the following information about currently running jobs and jobs waiting in the queue: Job ID, Partition, Job Name, User Name, Job Status, Run Time, Node Count, and Node List. There are a large number of command-line options available for customizing the information provided by `squeue` . Below are a list of examples:

Command	**Meaning**
`squeue --long`	Provide more job information
`squeue --user=USER_ID`	Provide information for `USER_ID`'s jobs
`squeue --account=ACCOUNT_ID`	Provide information for jobs running und
`squeue --states=running`	Show running jobs only
`squeue --Format=account,username,numcpus,state,timeleft`	Customize output of `squeue`
`squeue --start`	List estimated start time for queued jobs
`squeue --help`	Show all options

For more information about `squeue` see: http://slurm.schedmd.com/squeue.html

3.3 `sacct`

This command is used for viewing information for completed jobs. This can be useful for monitoring job progress or diagnosing problems that occurred during job execution. By default, `sacct` will report Job ID, Job Name, Partition, Account, Allocated CPU Cores, Job State, and Exit Code for all of the current user's jobs that completed since midnight of the current day. Many options are available for modifying the information output by `sacct` :

Command
`sacct --starttime 12.04.14`
`sacct --allusers`
`sacct --accounts=ACCOUNT_ID`
`sacct --format="JobID,user,account,elapsed, Timelimit,MaxRSS,ReqMem,MaxVMSize,ncpus,ExitCode`
`sacct --help`

The `--format` option is particularly useful, as it allows a user to customize output of job usage statistics. We would suggest create an alias for running a customized version of `sacct` . For instance, the `elapsed` and `Timelimit` arguments allow for a comparison of allocated vs. actual wall time. `MaxRSS` and `MaxVMSize` shows maximum RAM and virtual memory usage information for a job, respectively, while `ReqMem` reports the amount of RAM requested.

For more information about `sacct` see: http://slurm.schedmd.com/sacct.html

3.4 `scontrol`

`scontrol` is used for monitoring and modifying queued jobs. One of its most powerful options is the `scontrol show job` option, which is analogous to Torque's `checkjob` command. `scontrol` is also used for holding and releasing jobs. Below is a list of useful `scontrol` commands:

Command	Meaning
`scontrol show job JOB_ID`	Show information for queued or running job
`scontrol hold JOB_ID`	Place hold on job
`scontrol release JOB_ID`	Release hold on job
`scontrol show nodes`	Show hardware details for nodes on cluster
`scontrol update JobID=JOB_ID Timelimit=1-12:00:00`	Change wall time to 1 day 12 hours
`scontrol update dependency=JOB_ID`	Add job dependency so that job only starts afte
`scontrol --help`	Show all options

Please note that the time limit or memory of a job can only be adjust for pending jobs, not for running jobs.

For more information about `scontrol` see: http://slurm.schedmd.com/scontrol.html

3.5 `salloc`

The function of `salloc` is to launch an interactive job on compute nodes. This can be useful for troubleshooting/debugging a program or if a program requires user input. To launch an interactive job requesting 1 node, 2 CPU cores, and 1 hour of wall time, a user would type:

```
salloc --nodes=1 --ntasks=2 --time=1:00:00
```

This command will execute and then wait for the allocation to be obtained. Once the allocation is granted, an interactive shell is initiated on the allocated node (or one of the allocated nodes, if multiple nodes were allocated). At this point, a user can execute normal commands and launch his/her application like normal.

Note that many of the `sbatch` options are also applicable for `salloc` , so a user can insert other typical resource requests, such as memory. Another useful feature in `salloc` is that it enforces resource requests to prevent users or applications from using more resources than were requested. For example:

```
[bob@vmps12 ~]$ salloc --nodes=1 --ntasks=2 --time=1:00:00
salloc: Pending job allocation 1772833
salloc: job 1772833 queued and waiting for resources
salloc: job 1772833 has been allocated resources
salloc: Granted job allocation 1772833
[bob@vmp586 ~]$ hostname
vmp586
[bob@vmp586 ~]$ srun -n 2 hostname
vmp586
vmp586
[bob@vmp586 ~]$ srun -n 4 hostname
srun: error: Unable to create job step: More processors requested than permitted
[bob@vmp586 ~]$ exit
exit
srun: error: vmp586: task 0: Exited with exit code 1
salloc: Relinquishing job allocation 1772833
salloc: Job allocation 1772833 has been revoked.
[bob@vmps12 ~]$
```

In this example, `srun -n 4` failed because only 2 tasks were allocated for this interactive job (for details on `srun` see Section 3.9 below). Also note that typing `exit` during the interactive session will kill the interactive job, even if the allotted wall time has not been reached.

For more information about `salloc` see: http://slurm.schedmd.com/salloc.html

3.6 `xalloc`

Similarly to `salloc` , this command provides an interactive shell on a compute node but with the possibility of running programs with a graphical user interface (GUI) directly on the compute node. To correctly visualize the GUI on your monitor, you first need to connect to the cluster's gateway with the X11 forwarding abilitated as follows:

```
[bob@bobslaptop ~]$ ssh -X bob@login.accre.vanderibilt.edu
```

Then from the gateway request the interactive job with X11 forwarding as in the following example:

```
[bob@vmps12 ~]$ xalloc --nodes=1 --ntasks=2 --time=1:00:00
srun: job 12555243 queued and waiting for resources
srun: job 12555243 has been allocated resources
[bob@vmp586 ~]$
```

At this point when launching a GUI based software, the interface should appear on your monitor.

3.7 sinfo

`sinfo` allows users to view information about SLURM nodes and partitions. A partition is a set of nodes (usually a cluster) defined by the cluster administrator. Below are a few example uses of `sinfo` :

Command	**Meaning**
`sinfo --Nel`	Displays info in a node-oriented format
`sinfo --partition=gpu`	Get information about GPU nodes
`sinfo --states=IDLE`	Displays info about idle nodes
`sinfo --help`	Show all options

For more information about `sinfo` see: http://slurm.schedmd.com/sinfo.html

3.8 sreport

`sreport` is used for generating reports of job usage and cluster utilization. It queries the SLURM database to obtain this information. By default information will be shown for jobs run since midnight of the current day. Some examples:

Command	**Meaning**
`sreport cluster utilization`	Show cluster utilization report
`sreport user top`	Show top 10 cluster users based on tot
`sreport cluster AccountUtilizationByUser start=2014-12-01`	Show account usage per user dating ba
`sreport job sizesbyaccount PrintJobCount`	Show number of jobs run on a per-grou
`sreport --help`	Show all options

For more information about `sreport` see: http://slurm.schedmd.com/sreport.html

3.9 srun

This command is used to launch a parallel job step. Typically, srun is invoked from a SLURM job script to launch a MPI job (much in the same way that mpirun or mpiexec are used). More details about running MPI jobs within SLURM are provided below . Please note that your application must include MPI code in order to run in parallel across multiple CPU cores using srun . Invoking srun on a non-MPI command or executable will result in this program being independently run X times on each of the CPU cores in the allocation.

Alternatively, srun can be run directly from the command line on a gateway, in which case srun will first create a resource allocation for running the parallel job. The -n [CPU_CORES] option is passed to specify the number of CPU cores for launching the parallel job step. For example, running the following command from the command line will obtain an allocation consisting of 16 CPU cores and then run the command hostname across these cores:

```
srun -n 16 hostname
```

For more information about srun see: http://www.schedmd.com/slurmdocs/srun.html

4 ACCRE Commands

In addition to commands provided by SLURM, ACCRE staff have also written a number of useful commands that are available for use on the ACCRE cluster.

4.1 rtracejob

rtracejob is used to compare resource requests to resource usage for an individual job. It takes a job id as its single argument. For example:

```
[bob@vmps12 ~]$ rtracejob 1234567
+------------------+--------------------------+
|  User: bob       |      JobID: 1234567      |
+------------------+--------------------------+
| Account          | chemistry                |
| Job Name         | python.slurm             |
| State            | Completed                |
| Exit Code        | 0:0                      |
| Wall Time        | 00:10:00                 |
| Requested Memory | 1000Mc                   |
| Memory Used      | 13712K                   |
| CPUs Requested   | 1                        |
| CPUs Used        | 1                        |
| Nodes            | 1                        |
| Node List        | vmp505                   |
| Wait Time        | 0.4 minutes              |
| Run Time         | 0.4                      |
| Submit Time      | Thu Jun 18 09:23:32 2015 |
| Start Time       | Thu Jun 18 09:23:57 2015 |
| End Time         | Thu Jun 18 09:24:23 2015 |
+------------------+--------------------------+
| Today's Date     | Thu Jun 18 09:25:08 2015 |
+------------------+--------------------------+
```

`rtracejob` is useful for troubleshooting when something goes wrong with your job. For example, a user might want to check how much memory a job used compared to how much was requested, or how long it took a job to execute relative to how much wall time was requested. In this example, note the `Requested Memory` reported is 1000Mc, meaning 1000 megabytes per core (the "c" stands for "core"). This is the default for jobs that specify no memory requirement. If you see a lowercase "n" on the `Requested Memory` line, this stands for "node" and occurs when a `--mem=` line is included in a SLURM script, which allocates the amount of memory listed per node in the allocation.

4.2 q3

q3 is a useful command for getting a breakdown of currently running or recently run jobs and their states, organized by user, group, and account. The command takes no arguments and after a few seconds will produce output with a format similar to the following:

```
[jill@vmps12 ~]$ q3
+------------+------------+-------------+-----------+
| User       | Total Jobs | Total Cores | State     |
+------------+------------+-------------+-----------+
| jack       |     1      |      0      | Pending   |
| jack       |     7      |      7      | Running   |
| jack       |     59     |      59     | Completed |
| jack       |     2      |      2      | Failed    |
| jack       |     1      |      1      | Timed Out |
| jill       |     12     |      24     | Running   |
| jill       |     7      |      14     | Completed |
+------------+------------+-------------+-----------+
+--------------------+------------+-------------+-----------+
| Group              | Total Jobs | Total Cores | State     |
+--------------------+------------+-------------+-----------+
| science            |     1      |      0      | Pending   |
| science            |     19     |      31     | Running   |
| science            |     66     |      73     | Completed |
| science            |     2      |      2      | Failed    |
| science           |     1      |       1      | Timed Out |
+-------------------+------------+-------------+-----------+
+-----------------+-----------+-----------+--------------+--------------------+------+------
| Account         | Fairshare | Max Cores | Max Mem (MB) | Max CPU Time (Min) | Jobs | Cores |  State
+-----------------+-----------+-----------+--------------+--------------------+------+------
| science_account |    12     |    360    |   2457600    |      1382400       |  1   |   0   | Pending   |
| science_account |    12     |    360    |   2457600    |      1382400       |  19  |   31  | Running   |
| science_account |    12     |    360    |   2457600    |      1382400       |  66  |   73  | Completed |
| science_account |    12     |    360    |   2457600    |      1382400       |  2   |   2   | Failed    |
| science_account |    12     |    360    |   2457600    |      1382400       |  1   |   1   | Timed Out |
+-----------------+-----------+-----------+--------------+--------------------+------+------
+-------------+------+-------+
|             | Jobs | Cores |
+-------------+------+-------+
| Pending     |  1   |   0   |
| Running     |  19  |   31  |
| Completed   |  66  |   73  |
| Failed Jobs |  2   |   2   |
| Timed Out   |  1   |   1   |
+-------------+------+-------+
```

In this example, two users (jack and jill) are running jobs on the cluster. Both of these users are in a group called `science` , which is under an account called `science_account` . Accounts are important because resource limits are generally enforced on the account level, so `q3` makes it easy to compare an account's usage to its limits and to see which users are running jobs under an account. The three types of limits are Max Cores, Max Mem, and Max CPU Time, each of which limit the resources available to all jobs running under an account. For reference, if a job is pending due to a resource limitation, this will be indicated in the far right column from the output of `squeue` . AssocGrpCpuLimit, AssocGrpMemLimit, and AssocGrpRunMinsLimit are the reasons that will be shown by `squeue` based on limits on CPU cores, memory, or CPU time, respectively.

4.3 qSummary

`qSummary` provides an alternate summary of jobs and cores running across all groups in the cluster. It is possible to filter the results by selecting a specific account through the -g option.

```
[jill@vmps12 ~]$ qSummary
GROUP    USER     ACTIVE_JOBS  ACTIVE_CORES  PENDING_JOBS  PENDING_CORES
-----------------------------------------------------------------------------
science                   18           34             5            7
          jack             5            5             4            4
          jill            13           29             1            3
```

```
-----------------------------------------------------------------------------
economics                88          200           100           100
          emily          88          200           100           100
-----------------------------------------------------------------------------
Totals:                 106          234           105           107
```

As shown, the output from `qSummary` provides a basic view of the active and pending jobs and cores across groups and users within a group.

4.4 `showLimits`

As the name suggests, `showLimits` will display the resource limits imposed on accounts and groups on the cluster. Running the command without any arguments will list all accounts and groups on the cluster. Optionally, `showLimits` also accepts a -g argument followed by the name of a group or account. For example, to see a list of resource limits imposed on an account named `science_account` (this account does not actually exist on the cluster):

```
[jill@vmps12 ~]$ showLimits -g science_account
ACCOUNT      GROUP     FAIRSHARE  MAXCPUS  MAXMEM(GB)  MAXCPUTIME(HRS)
-----------------------------------------------------------------------------
science_account              12       3600       2400          23040
             biology         1        2400       1800              -
             chemistry        1        800        600              -
             physics         1         600        600           8640
             science         1           -       2200          20000
-----------------------------------------------------------------------------
```

Limits are always imposed on the account level, and occasionally on the group level when multiple groups fall under a single account. If a particular limit is not defined on the group level, the group is allowed access to the entire limit under its parent account. For example, the `science` group does not have a MAXCPUS limit defined, and therefore can run across a maximum of 3600 cores so long as no other groups under `science_account` are running and no other limits (`MAXMEM` or `MAXCPUTIME`) are exceeded.

We leave `FAIRSHARE` defined on the account level only, so groups within the same account do not receive elevated priority relative to one another. The value 1 for FAIRSHARE defined at the group level means that all groups under the account receive equal relative priority.

4.5 `SlurmActive`

`SlurmActive` displays a concise summary of the percentage of CPU cores and nodes currently allocated to jobs, and the number of memory-starved CPU cores on the cluster. For GPU accelerated nodes it will show the number of allocated GPUs.

```
[bob@vmps12 ~]$ SlurmActive
Standard Nodes Info:   589 of  589 nodes active               (100.00%)
                  5408 of 6008 processors in use by local jobs ( 90.01%)
                   461 of 6008 processors are memory-starved  (  7.67%)
                     139 of 6008 available processors         (  2.31%)

GPU Nodes Info:     Fermi:  32 of 64 GPUs in use              ( 50.00%)
                   Maxwell: 0 of 48 GPUs in use               (  0.00%)

Phi Nodes Info:     0 of  4 nodes active                      (  0.00%)
                 0 of 64 processors in use by local jobs      (  0.00%)
                 0 of 64 processors are memory-starved        (  0.00%)

ACCRE Cluster Totals:   597 of  621 nodes active               ( 96.14%)
                  5472 of 6344 processors in use by local jobs ( 86.25%)
                   461 of 6344 processors are memory-starved  (  7.27%)
                     411 of 6344 available processors         (  6.48%)

2079 running jobs, 7162 pending jobs
```

Multiple sections are reported. In general, the **Standard Node Info** section is the one users are most interested in, as this corresponds to the default "production" partition on the ACCRE cluster. **GPU Node Info** provides information about the availability of GPU nodes on the cluster, while the **Phi Node Info** section provides details about the availability of the Intel Xeon Phi nodes.

`SlurmActive` also reports the number of memory-starved cores in each section. A core is considered memory-starved if it is available for jobs but does not have access to at least 1GB of RAM (by default, jobs are allocated 1GB RAM per core). Requesting less than 1GB of RAM per core may provide access to these cores. Note that `SlurmActive` accepts a -m option followed by the amount of RAM (in GB) if you would like to compute memory-starved cores on the basis of another memory value. For example, `SlurmActive -m 2` will report cores as being memory-starved if they do not have access to at least 2GB of RAM.

5 Parallel Job Example Scripts

Below are example SLURM scripts for jobs employing parallel processing. More basic, non-parallel example scripts can be found in Section 2 , Section 3.1 , and in our Getting Started pages. In general, parallel jobs can be separated into four categories:

- Distributed memory programs that include explicit support for message passing between processes (e.g. MPI). These processes execute across multiple CPU cores and/or nodes.

- Multithreaded programs that include explicit support for shared memory processing via multiple threads of execution (e.g. Posix Threads or OpenMP) running across multiple CPU cores.
- Embarrassingly parallel analysis in which multiple instances of the same program execute on multiple data files simultaneously, with each instance running independently from others on its own allocated resources (i.e. CPUs and memory). SLURM job arrays offer a simple mechanism for achieving this.
- GPU (graphics processing unit) programs including explicit support for offloading to the device via languages like CUDA or OpenCL.

It is important to understand the capabilities and limitations of an application in order to fully leverage the parallel processing options available on the ACCRE cluster. For instance, many popular scientific computing languages like Python , R , and Matlab now offer packages that allow for GPU or multithreaded processing, especially for matrix and vector operations.

5.1 MPI Jobs

Jobs running MPI (Message Passing Interface) code require special attention within SLURM. SLURM allocates and launches MPI jobs differently depending on the version of MPI used (e.g. OpenMPI, MPICH2, Intel MPI). We recommend using OpenMPI version 1.8.4 (to load, type `setpkgs -a openmpi_1.8.4`) to compile code and then using SLURM's `srun` command to launch parallel MPI jobs. The example below runs MPI code compiled by OpenMPI 1.8.4:

```
#!/bin/bash
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --nodes=3
#SBATCH --tasks-per-node=8     # 8 MPI processes per node
#SBATCH --time=7-00:00:00
#SBATCH --mem=4G     # 4 GB RAM per node
#SBATCH --output=mpi_job_slurm.log
setpkgs -a openmpi_1.8.4
echo $SLURM_JOB_NODELIST
srun --mpi=pmi2 ./test  # srun is SLURM's version of mpirun/mpiexec
```

This example requests 3 nodes and 8 tasks (i.e. processes) per node, for a total of 24 MPI tasks. By default, SLURM allocates 1 CPU core per process, so this job will run across 24 CPU cores. Note that `srun` accepts many of the same arguments as `mpirun` / `mpiexec` (e.g. `-n <number cpus>`) but also allows increased flexibility for task affinity, memory, and many other features. Type `man srun` for a list of options. The `--mpi=pmi2` argument is required for MPI programs built with OpenMPI 1.8.4. Alternatively, MPI programs built with Intel's MPI (setpkgs -a intel_cluster_studio_compiler) do not require this additional argument.

Executables generated with older versions of OpenMPI or MPICH2 should be launched using these packages' native `mpirun` or `mpiexec` commands rather than SLURM's `srun` . Such programs may run under SLURM but in some cases they may not. In either case, we recommend updating to OpenMPI 1.8.4 as this library is built against SLURM and thus offers increased flexibility and reliability in our cluster environment.

More information about running MPI jobs within SLURM can be found here here: http://slurm.schedmd.com/mpi_guide.html Feel free to open a help desk ticket if you require assistance with your MPI job.

5.2 Multithreaded Jobs

Multithreaded programs are applications that are able to execute in parallel across multiple CPU cores within a single node using a shared memory execution model. In general, a multithreaded application uses a single process (i.e. "task" in SLURM) which then spawns multiple threads of execution. By default, SLURM allocates 1 CPU core per task. In order to make use of multiple CPU cores in a multithreaded program, one must include the `--cpus-per-task` option. The ACCRE cluster features 8-core and 12-core nodes, so a user can request up to 12 CPU cores per task. Below is an example of a multithreaded program requesting 4 CPU cores per task. The program itself is responsible for spawning the appropriate number of threads.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4  # 4 threads per task
#SBATCH --time=02:00:00   # two hours
#SBATCH --mem=4G
#SBATCH --output=multithread.out
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --job-name=multithreaded_example

# Run multi-threaded application
./hello
```

5.3 Job Arrays

Job arrays are useful for submitting and managing a large number of similar jobs. As an example, job arrays are convenient if a user wishes to run the same analysis on 100 different files. SLURM provides job array environment variables that allow multiple versions of input files to be easily referenced. In the example below , three input files called `vectorization_0.py` , `vectorization_1.py` , and `vectorization_2.py` are used as input for three independent Python jobs:

```
#!/bin/bash
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --ntasks=1
#SBATCH --time=2:00:00
#SBATCH --mem=2G
#SBATCH --array=0-2
#SBATCH --output=python_array_job_slurm_%A_%a.out

echo "SLURM_JOBID: " $SLURM_JOBID
echo "SLURM_ARRAY_TASK_ID: " $SLURM_ARRAY_TASK_ID
echo "SLURM_ARRAY_JOB_ID: " $SLURM_ARRAY_JOB_ID

setpkgs -a python2.7.8
python < vectorization_${SLURM_ARRAY_TASK_ID}.py
```

The `#SBATCH --array=0-2` line specifies the array size (3) and array indices (0, 1, and 2). These indices are referenced through the `SLURM_ARRAY_TASK_ID` environment variable in the final line of the SLURM batch script to independently analyze the three input files. Each Python instance will receive its own resource allocation; in this case, each instance is allocated 1 CPU core (and 1 node), 2 hours of wall time, and 2 GB of RAM.

One implication of allocating resources per task is that the node count will not apply across all tasks, so specifying `--nodes=1` will not limit all tasks within an array to a single node. To limit the total number of CPU cores (and thus tasks) used simultaneously, use `%[CPU_COUNT]` following the `--array=` option. For example, `--array=0-100%4` will limit the tasks to running on 4 CPU cores simultaneously. This means the tasks will execute in batches of 4 until all 100 tasks have completed.

The `--array=` option is flexible in terms of the index range and stride length. For instance, `--array=0-10:2` would give indices of 0, 2, 4, 6, 8, and 10.

The `%A` and `%a` variables provide a method for directing standard output to separate files. `%A` references the `SLURM_ARRAY_JOB_ID` while `%a` references `SLURM_ARRAY_TASK_ID`. SLURM treats job ID information for job arrays in the following way: each task within the array has the same `SLURM_ARRAY_JOB_ID`, and its own unique `SLURM_JOBID` and `SLURM_ARRAY_TASK_ID`. The JOBID shown from `squeue` is formatted by `SLURM_ARRAY_JOB_ID` followed by an underscore and the `SLURM_ARRAY_TASK_ID`.

While the previous example provides a relatively simple method for running analyses in parallel, it can at times be inconvenient to rename files so that they may be easy indexed from within a job array. The following example provides a method for analyzing files with arbitrary file names, provided they are all stored in a sub-directory named `data` :

```
#!/bin/bash
#SBATCH --mail-user=vunetid@vanderbilt.edu
#SBATCH --mail-type=ALL
#SBATCH --ntasks=1
#SBATCH --time=2:00:00
#SBATCH --mem=2G
#SBATCH --array=1-5   # In this example we have 5 files to analyze
#SBATCH --output=python_array_job_slurm_%A_%a.out
arrayfile=`ls data/ | awk -v line=$SLURM_ARRAY_TASK_ID '{if (NR == line) print $0}'`
setpkgs -a python2.7.8
python < data/$arrayfile
```

More information can be found here: http://slurm.schedmd.com/job_array.html

5.4 GPU Jobs

ACCRE has 30 compute nodes equipped with Nvidia GPU cards for general-purpose GPU computing. The nodes are divided into two partitions depending on the type of GPU available on the node:

`partition`	`fermi` ‘m	axwell‘
number of nodes	18	12
GPU	4 x GTX480	4 x GTX Titan X
CPU cores	8	12
CUDA cores (per GPU)	480	3072
host memory	48 GB	128 GB
GPU memory	1.5 GB	12 GB
network	10 Gbps Ethernet	56 Gbps RoCE
gres	1 GPU + 2 CPUs	1 GPU + 3 CPUs

Users can request the desired amount of GPUs by using SLURM generic resources, also called **gres** . Each gres bundles together one GPU to multiple CPU cores (see table above) belonging to the same PCI Express root complex to minimize data transfer latency between host and GPU memory. The number of CPU cores requested cannot be higher than the sum of cores in the requested gres .

Below is an example SLURM script header to request 2 GTX480 GPUs and 4 CPU cores on a single node on the `fermi` partition:

```
#!/bin/bash
#SBATCH --account=<your_gpu_account>
#SBATCH --partition=fermi
```

```
#SBATCH --gres=gpu:2
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --mem=20G
#SBATCH --time=2:00:00
#SBATCH --output=gpu-job.log
```

Note that you must be in one of the GPU groups on the cluster and specify this group from the job script in order to run jobs on the GPU cluster. The `#SBATCH --partition=<fermi OR maxwell>` line is also required in the job script.

Several versions of the Nvidia CUDA API are available on the cluster and can be selected via the `pkginfo` and `setpkgs` commands:

```
[bob@vmps12]$ pkginfo | grep cuda
                       cuda4.2   CUDA (4.2)
                       cuda5.0   CUDA (5.0)
                       cuda6.0   CUDA (6.0)
                       cuda6.5   CUDA (6.5)
                       cuda7.0   CUDA (7.0)
                       cuda7.5   CUDA (7.5)
[bob@vmps12]$ setpkgs -a cuda7.0
```

There are currently a handful applications available that allow you to leverage the low-latency RoCE network available on the `maxwell` partition. Note that both GPU partitions are intended for GPU jobs only, so users are not allowed to run purely CPU programs. The GPU nodes (both the `maxwell` and `fermi` partitions) support serial CPU execution as well as parallel CPU execution using either a multi-threaded, shared memory model (e.g. with OpenMP) or a multi-process, distributed memory execution (i.e. with MPI). Two flavors of RoCE-enabled MPI are available on the cluster, as well as Gromacs and HOOMD-Blue:

```
[bob@vmps12]$ pkginfo | grep _roce
  gromacs_5.1.2_roce  Gromacs with OpenMPI 1.10.2 for RoCE network and CUDA 7.5 support (GCC 4.
  hoomd_1.3.3_roce    HOOMD-Blue with OpenMPI 1.10.2 for RoCE network (GCC 4.9.3)
  mvapich2_2.1_roce   mvapich2 2.1 for RoCE network (GCC 4.9.3) [mpi]
  openmpi_1.10.2_roce OpenMPI 1.10.2 for RoCE network and CUDA 7.5 support (GCC 4.9.3) [mpi]
[bob@vmps12]$
```

All jobs making use of a RoCE-enabled MPI distribution should use SLURM's `srun` command rather than `mpirun/mpiexec` . Click here for an example of a HOOMD-Blue job.

In order to build a MPI application for the `maxwell` partition, we recommend launching an interactive job on one of the `maxwell` nodes via `salloc` :

```
salloc --partition=maxwell --account=<group> --gres=gpu:1 --time=4:00:00 --mem=20G
```

To test your application without submitting a batch job you can request an interactive job session via `salloc` as explained in the corresponding paragraphs.

This will not work with multiple GPU applications that require the use of `srun` .

It is possible to check the status of the GPU compute nodes by using the `gpustate` command:

```
[bob@vmps13]$ gpustate

==========================         ==========================
         Fermi                              Maxwell
==========================         ==========================
    Total nodes -- 16                  Total nodes -- 12

    Up ----------- 13                  Up ----------- 11
      Mixed ------ 0                     Mixed ------ 3
      Allocated -- 12                    Allocated -- 1
      Idle ------- 1                     Idle ------- 7
      Reserved---- 0                     Reserved---- 1
    Draining ----- 0                   Draining ----- 0
    Drained ------ 2                   Drained ------ 0
    Down --------- 1                   Down --------- 0
    Offline ------ 1                   Offline ------ 0

    Total GPUs --- 52                  Total GPUs --- 44
      Idle ------- 4                     Idle ------- 33
      Used ------- 48                    Used ------- 11
==========================         ==========================

vmp805   ALLOCATED       4         vmp1243  MIXED           4
vmp806   IDLE+DRAIN      0         vmp1244  MIXED           2
vmp807   ALLOCATED       4         vmp1245  IDLE            0
vmp808   ALLOCATED       4         vmp1246  IDLE            0
vmp813   IDLE            0         vmp1247  IDLE            0
vmp815   ALLOCATED       4         vmp1248  IDLE            0
vmp816   IDLE+DRAIN      0         vmp1249  IDLE            0
vmp818   ALLOCATED       4         vmp1250  ALLOCATED       4
vmp824   ALLOCATED       4         vmp1251  MIXED           1
vmp826   ALLOCATED       4         vmp1252  IDLE            0
vmp833   ALLOCATED       4         vmp1253  IDLE            0
vmp834   ALLOCATED       4         vmp1254  RESERVED        0
vmp836   ALLOCATED       4
vmp837   ALLOCATED       4
vmp838   ALLOCATED       4
vmp844   IDLE            0
```

6 Torque Wrappers

Torque wrappers are distributed with SLURM to ease the transition from Torque to SLURM. Wrappers are available for virtually all the common Torque commands, including `qsub` , `qstat` , `qdel` , `qhold` , `qrls`, and `pbsnodes` . These wrappers are designed to function in the same way as their Torque counterparts, with support for many of the same options and flags. Therefore, users may be able to run their old Torque scripts without converting them (or with minimal modifications) to SLURM syntax. These jobs will still be managed by SLURM, but to the user it will still "feel" like a Torque environment.

While the Torque wrappers should aid the transition from Torque to SLURM, in the long run we encourage users to convert their job scripts to SLURM. There are a number of reasons for converting to SLURM. The first reason is that native SLURM scripts offer increased flexibility and control over jobs. As the SLURM code base continues to expand, it is unlikely that the Torque wrappers will be fully supported and able to handle more advanced use cases. Troubleshooting and debugging of Torque scripts will also be more difficult.

7 SLURM Environment Variables

Variable	**Meaning**
`SLURM_JOBID` J	ob ID
`SLURM_SUBMIT_DIR` Jo	b submission directory
`SLURM_SUBMIT_HOST` Na	me of host from which job was submitted
`SLURM_JOB_NODELIST` Na	mes of nodes allocated to job
`SLURM_ARRAY_TASK_ID` Tas	k id within job array
`SLURM_JOB_CPUS_PER_NODE` CPU	cores per node allocated to job
`SLURM_NNODES` N	umber of nodes allocated to job

Each of these environment variables can be referenced from a SLURM batch script using the `$` symbol before the name of the variable (e.g. `echo $SLURM_JOBID`). A full list of SLURM environment variables can be found here: http://slurm.schedmd.com/sbatch.html#lbAF