Running Jobs on Perlmutter

New User Training
September 28, 2022

Muaaz Awan
Application Performance Group
Introduction
Slurm Job Scheduler

• Perlmutter uses Slurm as its job scheduler.

• **Slurm** is an open source tool that performs workload management and job scheduling.

• Slurm takes care of three key responsibilities:
  o Allocation of resources.
  o Executing and monitoring jobs.
  o Managing queue of the submitted jobs
Node Types

- Perlmutter has two partitions i.e. CPU and GPU.
- Jobs are run on compute nodes (CPU or GPU).
- Initially you are placed on a login node.
- Login nodes are shared by multiple users, any misuse can result in other’s being affected.
- Compute nodes are exclusive with the exception of CPU nodes in ‘shared’ qos.
Node Types

- Perlmutter has 1536 GPU Nodes and 3072 CPU nodes.
- Each GPU node has a 64 core AMD Milan CPU (7763) and 4 NVIDIA A100 GPUs.
- Each CPU node has two 64 core AMD Milan CPUs.
- Each Milan CPU core has two hardware threads.
Types of Jobs

- Perlmutter will be used by 7000+ users, it's important that everyone can get resources that they need.
- NERSC provides job queues based on quality of service required.
- Debugging jobs typically require small amount of resources for short periods.
- Sometimes an interactive node may be needed.
- Large production jobs.
- Large jobs that can be preempted.
Submitting Jobs

• Jobs can be submitted to queueing system through sbatch or salloc:
  o sbatch <my_job_script>
  o salloc <options>
• The above methods list details about resources needed for a job and for how long.
• For instance, below is a request for a cpu node to be allocated for 5 mins.
  o salloc -N 1 -C cpu -q debug -t 5 -A <project>
Submitting jobs

Login node:
- Submit batch jobs via sbatch or salloc

Head compute node:
- Runs commands in batch script
- Issues job launcher “srun” to start parallel jobs on all compute nodes (including itself)

*figure courtesy Helen (2020 NERSC Training)
Launching Jobs
Sample Batch Script

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=gpu
#SBATCH --jobname=myjob
#SBATCH --license=scratch,cfs

export OMP_NUM_THREADS=1
srun -n 64 <executable>
```

```bash
#!/bin/bash
#SBATCH -A mxxx
#SBATCH -q regular
#SBATCH -N 2
#SBATCH -t 12:00:00
#SBATCH -C gpu
#SBATCH -J myjob
#SBATCH -L scratch,cfs

export OMP_NUM_THREADS=1
srun -n 64 <executable>
```
Sample Batch Script

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=gpu
#SBATCH --jobname=myjob
#SBATCH --license scratch,cfs
export OMP_NUM_THREADS=1
srun -n 64 <executable>
```
Sample Batch Script for a GPU node

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=gpu
#SBATCH --jobname=myjob
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=2
#SBATCH --gpus-per-node=4

export OMP_NUM_THREADS=1
srun -n 128 <executable>

\[ c = 2 * \left(\frac{64}{k}\right) \]

where:
\( k = \text{ntasks-per-node} \)
# Sample Batch Script for a CPU node

```
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=cpu
#SBATCH --jobname=myjob
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=2

export OMP_NUM_THREADS=1
srun -n 256 <executable>
```

\[ c = 2 \times \left(\frac{128}{k}\right) \]

where:

\[ k = \text{ntasks-per-node} \]
Launch options and affinity

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=cpu
#SBATCH --jobname=myjob
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=2

export OMP_NUM_THREADS=1
srun -n 256 -cpu-bind=cores <executable>
```

- **cpu-bind** option can be used to better utilize the node.
- use `-cpu-bind=cores` options if total tasks are less than physical cores
- For more options check: `srun --cpu-bind=help`

\[ c = 2 \times \frac{128}{k} \]

where:
\[ k = \text{ntasks-per-node} \]
Launch options and affinity (GPUs)

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=gpu
#SBATCH --jobname=myjob
#SBATCH --ntasks-per-node=64
#SBATCH --cpus-per-task=2
#SBATCH --gpus-per-node=4

export OMP_NUM_THREADS=1
srun -n 128 --cpu-bind=cores --gpus-bind=closest <executable>
```

- By default all processes will have access to all GPUs.
- A round robin assignment does not guarantee affinity.
- To guarantee that closest GPU is assigned: `-gpus-bind=closest`
- More options can be explored in srun manual page.

\[ c = 2 \times \left(\frac{64}{k}\right) \]

where:
\[ k = \text{ntasks-per-node} \]
Launch options and affinity (GPUs)

`srun -n8 --cpu-bind=cores ./vec_add`

Rank 1/8 (PID:73658 on Core: 16) from nid003497 sees 4 GPUs, GPU assigned to me is: = 0000:41:00.0
Other 3 GPUs are:
**rank = 0: 0000:03:00.0 **
**rank = 2: 0000:81:00.0 **
**rank = 3: 0000:C1:00.0 **

Rank 5/8 (PID:73662 on Core: 17) from nid003497 sees 4 GPUs, GPU assigned to me is: = 0000:41:00.0
Other 3 GPUs are:
**rank = 0: 0000:03:00.0 **
**rank = 2: 0000:81:00.0 **
**rank = 3: 0000:C1:00.0 **

Rank 0/8 (PID:73657 on Core: 0) from nid003497 sees 4 GPUs, GPU assigned to me is: = 0000:03:00.0
Other 3 GPUs are:
**rank = 1: 0000:41:00.0 **
**rank = 2: 0000:81:00.0 **
**rank = 3: 0000:C1:00.0 **

Rank 2/8 (PID:73659 on Core: 32) from nid003497 sees 4 GPUs, GPU assigned to me is: = 0000:81:00.0
Other 3 GPUs are:
**rank = 0: 0000:03:00.0 **
**rank = 1: 0000:41:00.0 **
**rank = 3: 0000:C1:00.0 **
Launch options and affinity (GPUs)

```bash
srun -n8 --cpu-bind=cores --gpu-bind=closest ./vec_add
```

NUMA node(s): 4
NUMA node0 CPU(s): 0-15,64-79
NUMA node1 CPU(s): 16-31,80-95
NUMA node2 CPU(s): 32-47,96-111
NUMA node3 CPU(s): 48-63,112-127

Rank 1/8 (PID:74002 on Core: 16) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:81:00.0
Other 0 GPUs are:
Rank 3/8 (PID:74004 on Core: 48) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:03:00.0
Other 0 GPUs are:
Rank 5/8 (PID:74006 on Core: 17) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:81:00.0
Other 0 GPUs are:
Rank 7/8 (PID:74008 on Core: 49) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:03:00.0
Other 0 GPUs are:
Rank 0/8 (PID:74001 on Core: 0) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:C1:00.0
Other 0 GPUs are:
Rank 4/8 (PID:74005 on Core: 1) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:C1:00.0
Other 0 GPUs are:
Rank 6/8 (PID:74007 on Core: 33) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:41:00.0
Other 0 GPUs are:
Rank 2/8 (PID:74003 on Core: 32) from nid003497 sees 1 GPUs, GPU assigned to me is: = 0000:41:00.0
Thread Affinity

- Use OpenMP envs: OMP_PROC_BIND, OMP_PLACES to fine pin each thread to a subset of CPUs allocated to the host task

```bash
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=cpu
#SBATCH --jobname=myjob
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=2

export OMP_NUM_THREADS=2
export OMP_PLACES=threads
export OMP_PROC_BIND=true
srun -n 256 <executable>
```
Job Queues
Shared Queue

- Multiple jobs from different users can share a single node.
- This is advantageous if you want to run a small job but don’t want to burn a lot of node hours.
- A typical use case would be to run a serial job.

Some considerations:
- shared qos is only available for Perlmutter CPU nodes.
- for serial jobs it’s not recommended to use srun.
- smaller jobs with multiple ranks can also use shared qos
- more examples at:
  - https://docs.nersc.gov/jobs/examples/

```bash
#SBATCH -q shared
#SBATCH -t 60
#SBATCH --mem=4GB
#SBATCH -C cpu
#SBATCH -J my_job
./<executable>
```
Debug and Interactive Queues

- Small jobs can be run interactively for instance for debugging and compilation purposes.
- Special queues are available for these purposes.
- Debug
  - A maximum of 8 nodes for 30 mins can be requested:
    - `salloc -A mxxx -N 8 -q debug -C cpu -t 30`
- Interactive:
  - A maximum of 4 nodes for 4 hours can be requested:
    - `salloc -A mxxx -N4 -q interactive -C cpu -t 240`
  - More information:
    - [https://docs.nersc.gov/jobs/policy/#qos-limits-and-charges](https://docs.nersc.gov/jobs/policy/#qos-limits-and-charges)
Preempt Queue

- Jobs in this queue can be preempted in the favor of a higher priority job.
- Jobs can be requeued.
- Your application must have checkpoint restart capabilities to take advantage of this.
- A typical use case would be a job that requires very long time to complete (it may take very long for it to schedule without preempt queue).

```bash
#SBATCH -q preempt
#SBATCH -C gpu
#SBATCH -N 1
#SBATCH --time=24:00:00
#SBATCH --error=%x-%j.err
#SBATCH --output=%x-%j.out
#SBATCH --comment=96:00:00  #desired time limit
#SBATCH --signal=B:USR1@60  #sig_time (60 seconds) checkpoint overhead
#SBATCH --requeue
#SBATCH --open-mode=append
```
xfer Queue

- Configured for the purpose of **staging data from HPSS before run or archive result to HPSS after run**
- Helps avoid wasting compute hours.
- Can submit jobs to the xfer QOS from inside another batch script:
  - Add to the end of batch script: "sbatch --q xfer hsi put <my_file>
- [https://docs.nersc.gov/jobs/examples/#xfer-queue](https://docs.nersc.gov/jobs/examples/#xfer-queue)

```
#!/bin/bash
#SBATCH --qos=xfer
#SBATCH --time=12:00:00
#SBATCH --job-name=my_transfer
#SBATCH --licenses=SCRATCH

# Archive my_file to HPSS
htar -cvf my_file.tar my_file
```
Advanced Options
### Bundling Jobs

**Running Sequentially**

```bash
#SBATCH -q regular
#SBATCH -N 4
#SBATCH -t 6:00:00
#SBATCH -J my_job
#SBATCH -C cpu

srun -n 512 ./<exec_a>
srun -n 512 ./<exec_b>
srun -n 512 ./<exec_c>
```

**Running Concurrently**

```bash
#SBATCH -q regular
#SBATCH -N 6
#SBATCH -t 6:00:00
#SBATCH -J my_job
#SBATCH -C cpu

srun -N 2 -n 256 ./<exec_a> &
srun -N 2 -n 256 ./<exec_b> &
srun -N 2 -n 256 ./<exec_c> &
wait
```
Job Chaining

- Jobs can be scheduled to run dependent on the completion of other jobs.
- Runs only on successfully completion when `--dependency=afterok`
- Runs regardless of how the previous job ended: `--dependency=afterany`

% jobid1=$(sbatch --parsable first-job.sh)
% jobid2=$(sbatch --parsable --dependency=afterok:$jobid1 second-job.sh)
% jobid3=$(sbatch --parsable --dependency=afterok:$jobid2 third-job.sh)
% sbatch --dependency=afterok:$jobid2,afterok:$jobid3 fourth-job.sh

% jobid=$(sbatch --parsable first job.sh)
% sbatch --dependency=afterany :$jobid second_job.sh
Job Chaining

- Can also be used within an sbatch script

```
#!/bin/bash
#SBATCH --account=mxxx
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=cpu
#SBATCH --jobname=myjob
#SBATCH --ntasks-per-node=128
#SBATCH --cpus-per-task=2
#SBATCH --dependency=afterok:<jobid>

export OMP_NUM_THREADS=1
srun -n 256 <executable>
```
Job Arrays

#!/bin/bash
#SBATCH --account=mxxx_g
#SBATCH --qos=regular
#SBATCH --nodes=2
#SBATCH --time=60
#SBATCH --constraint=gpu
#SBATCH --jobname=myjob
#SBATCH --license scratch,cfs
#SBATCH --array=1-10

cd job_${SLUR_ARRAY_JOB_ID}
srun -n 64 <executable>

- a nice way of managing multiple jobs
- doesn't necessarily offer good turn around time.
NERSC offers a job script generator. It can help generate a basic script with some friendly comments. Users may need to make minor changes in the generated scripts. https://my.nersc.gov/script_generator.php
Multi Process Service (GPU nodes)

- NVIDIA’s multi process service (MPS) allows sharing a GPU across multiple processes.
- This can help oversubscribe GPUs for better performance overall.
- MPS needs to be explicitly enabled on all the nodes running the job.
- MPS can be enabled using:
  - `nvidia-cuda-mps-control -d`
- And disabled using:
  - `echo quit | nvidia-cuda-mps-control`
- It is recommended that the NERSC provided wrapper script be used if you plan on using MPS with your application. Wrapper script can be found at:
- Before using MPS make sure that all the GPUs are visible to all the ranks (refer to the GPU affinity portion)
Monitoring Jobs
Monitoring Your Jobs

- Once your job is submitted, it enters the queue and will start when resources are available.
- Overall job priorities are a combination of QOS, queue wait time, job size, wall time request.
- Jobs can be monitored using:
  - `sqs`
  - `squeue`
  - `sacct`
Using squeue and sqs

**squeue**
- By default squeue displays jobs from all the users.
- `squeue -u <user id>` displays job for a particular user
- More options can be explored on the manual pages.

**sqs**
- `sqs` is a NERSC wrapper on squeue.
- By default `sqs` displays jobs from current user
- More options can be explored using `sqs --help`
**sacct**

- Provides job accounting information about active and completed jobs
- Maximum query duration is one month.

```bash
mgawan@perlmutter:login29:~> sacct -u mgawan -S 2022-08-25 -E 2022-08-30

<table>
<thead>
<tr>
<th>JobID</th>
<th>JobName</th>
<th>Partition</th>
<th>Account</th>
<th>AllocCPUS</th>
<th>State</th>
<th>ExitCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>3046996</td>
<td>xlarge.sh</td>
<td>regular_m+</td>
<td>m888</td>
<td>86528</td>
<td>FAILED</td>
<td>15:0</td>
</tr>
<tr>
<td>3046996</td>
<td>batch</td>
<td>m888</td>
<td></td>
<td>256</td>
<td>FAILED</td>
<td>15:0</td>
</tr>
<tr>
<td>3046996</td>
<td>extern</td>
<td>m888</td>
<td></td>
<td>86528</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>3046996</td>
<td>toast_ben+</td>
<td>m888</td>
<td></td>
<td>86528</td>
<td>CANCELLED</td>
<td>0:15</td>
</tr>
<tr>
<td>3052441</td>
<td>xlarge.sh</td>
<td>regular_m+</td>
<td>m888</td>
<td>86528</td>
<td>FAILED</td>
<td>15:0</td>
</tr>
<tr>
<td>3052441</td>
<td>batch</td>
<td>m888</td>
<td></td>
<td>256</td>
<td>FAILED</td>
<td>15:0</td>
</tr>
<tr>
<td>3052441</td>
<td>extern</td>
<td>m888</td>
<td></td>
<td>86528</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>3052441</td>
<td>toast_ben+</td>
<td>m888</td>
<td></td>
<td>86528</td>
<td>CANCELLED</td>
<td>0:15</td>
</tr>
<tr>
<td>3052445</td>
<td>xlarge.sh</td>
<td>regular_m+</td>
<td>m888</td>
<td>86528</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>3052445</td>
<td>batch</td>
<td>m888</td>
<td></td>
<td>256</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
<tr>
<td>3052445</td>
<td>extern</td>
<td>m888</td>
<td></td>
<td>86528</td>
<td>COMPLETED</td>
<td>0:0</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>User</th>
<th>JobID</th>
<th>Elapsed</th>
<th>NNodes</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>mgawan</td>
<td>3046996</td>
<td>00:10:06</td>
<td>338</td>
<td>FAILED</td>
</tr>
<tr>
<td></td>
<td>3046996.bat+</td>
<td>00:10:06</td>
<td>1</td>
<td>FAILED</td>
</tr>
<tr>
<td></td>
<td>3046996.ext+</td>
<td>00:10:06</td>
<td>338</td>
<td>COMPLETED</td>
</tr>
<tr>
<td></td>
<td>3046996.0</td>
<td>00:10:00</td>
<td>338</td>
<td>CANCELLED</td>
</tr>
<tr>
<td>mgawan</td>
<td>3052441</td>
<td>00:08:00</td>
<td>338</td>
<td>FAILED</td>
</tr>
<tr>
<td></td>
<td>3052441.bat+</td>
<td>00:08:00</td>
<td>1</td>
<td>FAILED</td>
</tr>
<tr>
<td></td>
<td>3052441.ext+</td>
<td>00:08:00</td>
<td>338</td>
<td>COMPLETED</td>
</tr>
<tr>
<td></td>
<td>3052441.0</td>
<td>00:07:58</td>
<td>338</td>
<td>CANCELLED</td>
</tr>
<tr>
<td>mgawan</td>
<td>3052445</td>
<td>00:25:44</td>
<td>338</td>
<td>COMPLETED</td>
</tr>
<tr>
<td></td>
<td>3052445.bat+</td>
<td>00:25:44</td>
<td>1</td>
<td>COMPLETED</td>
</tr>
<tr>
<td></td>
<td>3052445.ext+</td>
<td>00:25:45</td>
<td>338</td>
<td>COMPLETED</td>
</tr>
<tr>
<td></td>
<td>3052445.0</td>
<td>00:25:39</td>
<td>338</td>
<td>COMPLETED</td>
</tr>
</tbody>
</table>
scontrol

- To view job details such as start end time, resources used you can use:
  - `scontrol show job <jobid>`

- To update certain job specifications after submitting
  - `scontrol update <SPECIFICATION>`
mgawan@perlmutter:login36:/> scontrol show job 3247429
JobId=3247429  JobName=large_128.sh
 UserId=mgawan(80008)  GroupId=mgawan(80008)  MCS_label=N/A
  Priority=67679  Nice=0  Account=m888_g  QOS=gpu_regular
  JobState=PENDING  Reason=Priority  Dependency=(null)
  Requeue=0  Restart=0  BatchFlag=1  Reboot=0  ExitCode=0:0
  RunTime=00:00:00  TimeLimit=01:10:00  TimeMin=N/A
  SubmitTime=2022-09-23T23:06:00  EligibleTime=2022-09-23T23:06:00
  AccrueTime=2022-09-23T23:06:01
  StartTime=Unknown  EndTime=Unknown  Deadline=N/A
  SuspendTime=None  SecsPreSuspend=0  LastSchedEval=2022-09-23T23:06:07
  Scheduler=Main
  Partition=gpu_ss11  AllocNode:Sid=login36:86362
  ReqNodeList=(null)  ExcNodeList=(null)
  NodeList=(null)
  NumNodes=128-128  NumCPUs=16384  NumTasks=8192
  CPUs/Task=2  ReqB:S:C:T=0:0:*:*
  TRES=cpu=16384,mem=28750G,node=128,billing=16384
  Socks/Node=*  NtasksPerN:B:S:C=64:0:*:*
  CoreSpec=*  MinCPUsNode=128  MinMemoryNode=0  MinTmpDiskNode=0
  Features=gpu  DelayBoot=00:00:00
  OverSubscribe=NO  Contiguous=0  Licenses=cfs:1
  Command=/global/cfs/cdirs/nstaff/mgawan/perlmutter_gpu_nodes/large/large_128.sh
  WorkDir=/global/cfs/cdirs/nstaff/mgawan/perlmutter_gpu_nodes/large
  StdErr=/global/cfs/cdirs/nstaff/mgawan/perlmutter_gpu_nodes/large/slurm-3247429.out
  StdIn=/dev/null
  StdOut=/global/cfs/cdirs/nstaff/mgawan/perlmutter_gpu_nodes/large/slurm-3247429.out
  Power=
  TresFreq=gpu:high
mgawan@perlmutter:login36:/> sqs

<table>
<thead>
<tr>
<th>JOBID</th>
<th>ST</th>
<th>USER</th>
<th>NAME</th>
<th>NODES</th>
<th>TIME_LIMIT</th>
<th>TIME</th>
<th>SUBMIT_TIME</th>
<th>QOS</th>
<th>START_TIME</th>
<th>FEATURES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3247429</td>
<td>PD</td>
<td>mgawan</td>
<td>large_128.sh</td>
<td>128</td>
<td>1:10:00</td>
<td>0:00</td>
<td>2022-09-23T23:06:00</td>
<td>gpu_regular</td>
<td>N/A</td>
<td>gpu</td>
<td>(None)</td>
</tr>
</tbody>
</table>

mgawan@perlmutter:login36:/> scontrol update jobid=3247429 qos=early_science

mgawan@perlmutter:login36:/> sqs

<table>
<thead>
<tr>
<th>JOBID</th>
<th>ST</th>
<th>USER</th>
<th>NAME</th>
<th>NODES</th>
<th>TIME_LIMIT</th>
<th>TIME</th>
<th>SUBMIT_TIME</th>
<th>QOS</th>
<th>START_TIME</th>
<th>FEATURES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3247429</td>
<td>PD</td>
<td>mgawan</td>
<td>large_128.sh</td>
<td>128</td>
<td>1:10:00</td>
<td>0:00</td>
<td>2022-09-23T23:06:00</td>
<td>early_science</td>
<td>N/A</td>
<td>gpu</td>
<td>(None)</td>
</tr>
</tbody>
</table>
scancel

- For cancelling the scheduled jobs:
  - `scancel <jobid>`

```bash
mgawan@perlmutter:login36:/> scancel 3247429
mgawan@perlmutter:login36:/> sqs
```

### Table

<table>
<thead>
<tr>
<th>JOBID</th>
<th>ST</th>
<th>USER</th>
<th>NAME</th>
<th>NODES</th>
<th>TIME_LIMIT</th>
<th>TIME</th>
<th>SUBMIT_TIME</th>
<th>QOS</th>
<th>START_TIME</th>
<th>FEATURE</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
</table>

---

[NERSC](https://www.nersc.gov)  | [BERKELEY LAB](https://www.lbl.gov)  | [U.S. DEPARTMENT OF ENERGY](https://www.energy.gov)  | Office of Science
Best Practices
# Perlmutter GPU Queue Policy (as of Sept. 2022)

## Perlmutter GPU

<table>
<thead>
<tr>
<th>QOS</th>
<th>Max nodes</th>
<th>Max time (hrs)</th>
<th>Submit limit</th>
<th>Run limit</th>
<th>Priority</th>
<th>QOS Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>128</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>interactive</td>
<td>4</td>
<td>4</td>
<td>5000</td>
<td>2</td>
<td>high</td>
<td>-</td>
</tr>
<tr>
<td>jupyter</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>high</td>
<td>-</td>
</tr>
<tr>
<td>debug</td>
<td>8</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>preempt</td>
<td>128</td>
<td>24 (preemptible after two hours)</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>early_science</td>
<td>-</td>
<td>12</td>
<td>5000</td>
<td>10</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>realtime</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>very high</td>
<td>-</td>
</tr>
</tbody>
</table>
# Perlmutter CPU Queue Policy (as of Sept. 2022)

## Perlmutter CPU Queue Policy

<table>
<thead>
<tr>
<th>QOS</th>
<th>Max nodes</th>
<th>Max time (hrs)</th>
<th>Submit limit</th>
<th>Run limit</th>
<th>Priority</th>
<th>QOS Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>-</td>
<td>12</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>interactive</td>
<td>4</td>
<td>4</td>
<td>5000</td>
<td>2</td>
<td>high</td>
<td>-</td>
</tr>
<tr>
<td>jupyter</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>high</td>
<td>-</td>
</tr>
<tr>
<td>debug</td>
<td>8</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>shared</td>
<td>0.5</td>
<td>6</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>preempt</td>
<td>128</td>
<td>24 (preemptible after two hours)</td>
<td>5000</td>
<td>-</td>
<td>medium</td>
<td>-</td>
</tr>
<tr>
<td>realtime</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>custom</td>
<td>very high</td>
<td>-</td>
</tr>
</tbody>
</table>
Jobs Scheduling

- Each job has its priority value, composed of qos, job age, and a small value of fairshare.
- There are two Slurm schedulers: main and backfill.
- Every few minutes, the main scheduler schedules jobs in the order of priority list for a few days into the future.
- The backfill scheduler then schedules small and short jobs to run if they will not affect the start time of those jobs that are already scheduled by the main scheduler.
Tips for Getting Better Throughput

● Submit shorter jobs, they are easier to schedule
  ○ Checkpoint to break up long jobs, use variable time
  ○ Short jobs can take advantage of “backfill” opportunities

● Make sure the wall clock time you request is accurate
  ○ Larger shorter jobs are easier to schedule than long smaller jobs
  ○ Many users unnecessarily request the largest wall clock time possible as default
Large Jobs Considerations

- sbcast your executable to compute nodes before srun:
  
  ```
  sbcast --compress=lz4 /path/to/exe /tmp/exe
  srun /tmp/exe
  ```

  https://docs.nersc.gov/jobs/best-practices/#large-jobs

- Consider to use shifter for large jobs using shared libraries.
Other Running Jobs Considerations

- Remember to compile separately for each type of compute nodes
- **Running jobs from global homes is strongly discouraged**
  - IO is not optimized
  - The global homes file system access on compute nodes is much slower than from $SCRATCH
  - It may also cause negative impact for other users interactive response on the system
- Consider to put your project’s shared software in `/global/common/software/<project>`
  - It is mounted read-only on compute nodes, so has less impact than other GPFS file systems (global homes or community file system)
- Consider to adopt workflow tools for better managing your jobs
Further Information

- Best resource for all the information is NERSC’s documentation.
- Everything covered in these slides can be found in more details at:
  - [https://docs.nersc.gov/systems/perlmutter/running-jobs/](https://docs.nersc.gov/systems/perlmutter/running-jobs/)

- For any help please open a service ticket via Help Portal
  - [https://help.nersc.gov](https://help.nersc.gov)
Thank You and Welcome to NERSC!