Optimizing Resource Usage in Scheduled Jobs

Brian Vanderwende
CISL Consulting Services

November 20, 2020

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

A selective summary of our schedulable resources

Cheyenne

- CPU-only nodes
- 36 physical cores/tasks
- 72 SMT threads
- Regular nodes with 45 GB usable memory (default)
- Large mem nodes with 109 GB usable memory
- Exclusive or shared queues

Casper

- CPU & CPU+GPU nodes
  - GP100 and V100 GPUs
- 36 physical cores/tasks
- 72 SMT threads
- Node memory up to ~1100 GB
- Up to 32 GB memory per GPU
- Node “features” like X11, CPU types
  - SkyLake, CascadeLake CPUs

PBS Pro scheduling

Slurm scheduling...

Casper will be transitioned to PBS Pro in 2021

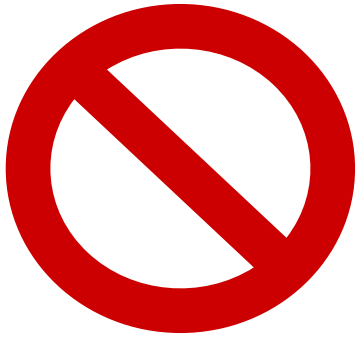
Why focus on resource requests?

Good resource requests can:

- Use less core hours from your allocation
- Increase job priority (lower queue times)
- Improve application performance

Bad resource requests can:

- Reach out-of-memory condition on Casper, leading to use of slower NVMe swap space
- Cause submit and runtime failures
- In rare cases, crash nodes



Resource requests in an MPI batch job on Cheyenne

```
#!/bin/bash
#PBS -A PROJ0001
#PBS -N mpi_job
#PBS -j oe
#PBS -k eod
#PBS -q regular
#PBS -l walltime=04:00:00
#PBS -l select=1:ncpus=30:mpiprocs=30:mem=109GB+4:ncpus=36:mpiprocs=36

### Application temp data to scratch
export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run MPI program
mpiexec_mpt ./app

### Store job statistics in log file
qstat -f $PBS_JOBID
```

- We expect application to have higher memory requirements on head node
- Implicit resource request of temp files on scratch (/tmp on nodes is very small!)
- Job statistics include CPU and memory utilization (purged after one day)

Job output and error log placement in PBS Pro

Application output and error messages are written to log files during runtime. There are three possible storage behaviors depending on configuration:

```
#!/bin/bash
#PBS -N mpi_job
#PBS -j oe
```

Default - log(s) are written to /tmp (memory) during runtime and then when the job completes they are moved to job directory on GLADE.

```
#!/bin/bash
#PBS -N mpi_job
#PBS -k oe
```

Home - log(s) are written to user's home directory during runtime and stay there after job completion. *Note that PBS does not respect the -j, -o, and -e options when using this mode.*

```
#!/bin/bash
#PBS -N mpi_job
#PBS -j oe
#PBS -k oed
```

Recommended - log(s) are written directly to the job directory on GLADE during runtime. Avoids exhausting memory or home directory space when jobs produce large quantities of logging messages.

Resource requests in an AI/ML GPU job on Casper

```
#!/bin/bash -l
#SBATCH --job-name=ML_job
#SBATCH --gres=gpu:v100:1
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=4
#SBATCH --mem=40G
#SBATCH --time=04:00:00
#SBATCH --account=PROJ0001
#SBATCH --partition=dav
#SBATCH --output=ml.out.%j

### Application temp data to scratch
export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

module load python
ncar_pylib

# Run machine learning driver script
python ml_driver.py

### Store job statistics in log file
scontrol show job $SLURM_JOBID
```

- We reserve a single V100 for GPU-enabled machine learning job
- Common to use 4-8 CPU tasks to manage data for a single GPU in AI/ML
- V100 has 32 GB of video memory - we request 40 GB of node memory to buffer data movement to and from GPU
- Again, small /tmp space on node so use GLADE scratch for temp files
- Store scontrol output including CPU and memory usage - purged quickly

Resource allocation and the shell in Slurm interactive jobs

execdav provides an interactive session on Casper via a two-step process

1) Use **salloc** to provision resources to a Slurm job
2) Use **srun** to distribute some/all of those resources to a bash/tcsh shell

By default, all memory and GPUs are allocated to the shell, leaving none for sub-jobs (programs run using **srun**). To get around this, specify shell resources after a separator.

```
# Distribute all resources to the shell (bash in this case)
cheyenne01$ execdav -n 4 --mem=10G -t 30 --gres=gpu:v100:1

# Only distribute small subset of resources to the shell
cheyenne01$ execdav -n 4 --mem=10G -t 30 --gres=gpu:v100:1 -- --mem=1G --gres=none
```

Resource complexities with an MPI-GPU Jacobi solver

Experiment 1

Request 2 tasks and 2 GPUs, allocating all resources to the interactive bash shell

```
cheyenne01$ execdav -t 10 -n 2 --mem=10G --gres=gpu:v100:2

# Start application using Slurm launcher
casper29$ srun -I ./jacobi
srun: error: Unable to create step for job 6202531: Requested nodes are busy

# Start application using Open MPI launcher
casper29$ mpirun ./jacobi
7168x7168: 1 GPU:   6.3649 s, 2 GPUs:  70.4340 s, speedup:     0.09, efficiency:     4.52
```

Resource complexities with an MPI-GPU Jacobi solver

Experiment 2

Request 4 tasks and 2 GPUs, allocating all resources to the interactive bash shell

```
cheyenne01$ execdav -t 10 -n 4 --mem=10G --gres=gpu:v100:2

# Start application using Open MPI launcher
casper29$ mpirun ./jacobi
7168x7168: 1 GPU:  13.6105 s, 4 GPUs: 141.2820 s, speedup:     0.10, efficiency:     2.41

# Only use two CPU tasks to drive application (avoid stacking work on GPUs)
casper29$ mpirun -n 2 ./jacobi
7168x7168: 1 GPU:   3.3795 s, 2 GPUs:   1.7077 s, speedup:     1.98, efficiency:    98.95
```

Resource complexities with an MPI-GPU Jacobi solver

Experiment 3

Request 4 tasks and 2 GPUs, but reserve most memory and both GPUs for srun sub-jobs

```
cheyenne01$ execdav -t 10 -n 4 --mem=11G --gres=gpu:v100:2 -- --mem=1G --gres=none

# Start application using Slurm launcher
casper29$ srun --mem=10G --gres=gpu:v100:2 -I ./jacobi
7168x7168: 1 GPU:  11.4907 s, 4 GPUs:   3.0765 s, speedup:     3.74, efficiency:    93.38

# Only use two CPU tasks to drive application (avoid stacking work on GPUs)
casper29$ srun -n 2 --mem=10G --gres=gpu:v100:2 -I ./jacobi
7168x7168: 1 GPU:   3.3928 s, 2 GPUs:   1.7074 s, speedup:     1.99, efficiency:    99.36

# Start application using Open MPI launcher
casper29$ mpirun -n 2 ./jacobi
mpirun noticed that process rank 1 with PID 17220 on node casper29 exited on signal 11
(Segmentation fault).
```

Query past PBS/Cheyenne jobs using “qhist”

CISL provides a custom utility, called **qhist**, which allows for queries into the PBS historical job database. You can:

- Search for all of your jobs over a date range or look back N days
- Display the average CPU and memory footprint per node for each job
- Filter to show only jobs that returned a non-zero (failure) status

```
cheyenne01$ qhist -u $USER -p 20201001-20201031 -s memory -t minutes | head -6
Job ID    User      Queue    Nodes   Submit   Start    Finish   Mem(GB)   CPU(%)   Wall(m)
4801178   vanderw   regular      4   29-1131  29-1146  29-1202     49.2     10.1     16.43
4805336   vanderw   regular      4   29-1320  29-1320  29-1328     42.6     32.8      7.52
4805722   vanderw   regular      4   29-1407  29-1410  29-1434     39.0     30.7     24.35
4703912   vanderw   regular      4   23-1835  23-1836  23-1930     33.8     95.2     54.68
4719620   vanderw   regular      4   25-1936  25-1940  25-2007     33.8     62.5     27.38
```

Query past Slurm jobs using “sacct” command

Slurm’s historical job database can be queried using **sacct**. You can:

- View your jobs over a given date range
- Customize the output fields to view only desired information

Slurm has concept of “job steps”, so resource usage typically in step JOBID.0

```
casper-login1$ export SACCT_FORMAT=jobid,user,end%20,elapsed%10,ncpus%4,nnodes%4,reqmem%7,maxrss,state
casper-login1$ sacct -u vanderwb -S 2020-08-01 -E 2020-08-31 --units=G | head -n 8
       JobID      User                  End    Elapsed NCPU NNod  ReqMem     MaxRSS      State
------------ --------- -------------------- ---------- ---- ---- ------- ---------- ----------
5668870       vanderwb  2020-08-03T13:33:44   00:05:06    2    1  1.78Gc             COMPLETED
5668870.ext+            2020-08-03T13:33:45   00:05:07    2    1  1.78Gc       0.00G COMPLETED
5668870.0               2020-08-03T13:33:44   00:05:03    1    1  1.78Gc       3.57G COMPLETED
5668884       vanderwb  2020-08-03T13:44:57   00:10:49    2    1    40Gn             COMPLETED
5668884.ext+            2020-08-03T13:44:58   00:10:50    2    1    40Gn       0.00G COMPLETED
5668884.0               2020-08-03T13:44:57   00:10:46    1    1    40Gn       6.73G COMPLETED
```

Monitoring runtime resource usage on a single node

top - get CPU and memory usage of all processes on a node

```
casper08$ top -b -n1 -u vanderwb -o +%MEM
top - 22:54:23 up 33 days,  4:44,  0 users,  load average: 7.61, 7.55, 6.90
Tasks: 766 total,   5 running, 761 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.2 us,  2.4 sy,  0.0 ni, 95.2 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem : 79102374+total, 73400243+free, 47664012 used,  9357316 buff/cache
KiB Swap: 19535134+total, 19203357+free, 33177836 used. 73731980+avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
152664 vanderwb  20   0   10.7g 674188 591916 R  94.7  0.1   0:01.17 jacobi
152666 vanderwb  20   0   10.8g 672200 591944 R  89.5  0.1   0:01.17 jacobi
```

nvidia-smi - get information about GPU specs and usage

```
casper08$ nvidia-smi pmon -c 1
# gpu        pid  type    sm   mem   enc   dec   command
# Idx          #   C/G     %     %     %     %   name
    0     152666     C    31     5     -     -   jacobi
    1     152664     C    30     5     -     -   jacobi
```

Accessing running jobs to analyze resource usage

For the duration of your job, you have SSH privileges to access the scheduled compute nodes. Use the **-n** option to qstat to get node information. (squeue provides a NODELIST field by default)

Once you have a node, simply SSH to it from one of the login nodes. Note that once your job ends, your session on the compute node will be terminated!

```
cheyenne1$ qstat -n -u vanderwb
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
5103836.chadmin vanderwb regular  STDIN       55037   1   1    --  01:00 R 00:00 r9i4n10/0

cheyenne1$ ssh r9i4n10
vanderwb@r9i4n10:~>
```

Monitoring runtime resource usage across multiple nodes

peak_memusage - provides maximum memory usage for each task in program (node memory only)

```
Used memory in task 1: 544.11MiB (+3.67MiB overhead). ExitStatus: 0. Signal: 0
Used memory in task 0: 542.12MiB (+3.67MiB overhead). ExitStatus: 0. Signal: 0
```

nvprof - profile many aspects of GPU performance and store profiles for subsequent analysis (*replaced soon by NSight*)

Arm MAP - extensive performance profiler that supports both CPU and GPU codes

Recognizing resource-related job failures

Sometimes you will see clues in the job logs that can motivate more investigation:

- MPT ERROR: Rank 0(g:0) received signal SIGBUS(7)...
  - Memory access error - typically indicates exhausting available RAM
- line 36: 16786 Killed mpirun…
  - Indicates the job was killed by an external watchdog; if walltime limit was not reached then also likely an out-of-memory failure
- CUDA error: out of memory
  - Application ran out of GPU memory (node memory limit may be fine)

If job fails with no logs, you may have run out of /tmp or quota on GLADE!

Common job resource pitfalls

- Request too little memory on Casper
  - Some data destined for RAM will end up in swap space, significantly impacting performance
- Requesting too much memory on Casper
  - Longer queue wait; wasted node resources
- Request multiple nodes but application does not support MPI or other distribution method
  - No means to split problem across nodes = resource waste
  - Can be easy to miss using GPU frameworks

General tips for resource scheduling

- Use the fewest resources required to safely run your application to optimize core-hour usage and queue wait times
- Only restrict job to specific node types if necessary for job execution… more flexible requests schedule faster
- **Verify that you are using what you think you are using!**

If running a job configuration for the first time...

1. Use conservative resource requests
2. Check what your job actually used
3. Lower resource requests as appropriate

Getting assistance from the CISL Help Desk

https://www2.cisl.ucar.edu/user-support/getting-help

- ~~Walk-in: ML 1B Suite 55~~
- Web: http://support.ucar.edu
- Phone: 303-497-2400

Specific questions from today and/or feedback:

- Email: vanderwb@ucar.edu