

ADVANCED USE OF SUPERCOMPUTERS

Imad Kissami

Imad.kissami@um6p.ma

2023/2024

Environment & Resources

● Environment

- Operating system : Unix/Linux
- Distribution : Redhat x86_64 server, 7.1
- Administration solution and deployment : BRIGHT cluster Manager (HPC x24 licenses)

● Resources

- Graphic cards (GPU) :
 - 7x Tesla Pascal 40 (GPU/server) : Calculus & IA
 - 5x Tesla VOLTA 100 (GPU/server) : Calculus & IA
- Compute nodes (696 cores):
 - 14 Skylake PowerEdge R740, 22C/CPU, 2 sockets, Intel(R) Xeon(R) Gold 6152 CPU@ 2.10GHz (386GB),
 - 2 Broadwell PowerEdge R430, 20C/CPU, 2 sockets, Intel(R) Xeon(R) CPU E5-2698 v4@ 2.20GHz (128GB),
- Many compilers and modules are available.

Simlab Storage

- Scratch/Parallel File Storage :
 - 2 servers Beegfs RAID 6, 148 TB; for temporary space (cleaned every 90 days)
- ISILON 1Po (outside of the cluster):
 - /home (18TB); space of calculation code and test files ($\sim 1\text{Mo}$)
 - /project (100TB); research teams can ask for Max of 5To (can be more)
- Local disks (\tmp, \local) (min 1,2 To,max 10,2 TB)

Module commands

- Displaying the installed modules:

```
$ module avail
----- /cm/local/modulefiles -----
cluster-tools-dell/8.1 freeipmi/1.5.7   lua/5.3.4   openldap
cluster-tools/8.1      gcc/7.2.0       module-git openmpi/mlnx/gcc/64/3.1.1rc1
cmd                    intel/mic/sdk/3.8.4 module-info shared
dot                    ipmitool/1.8.18   null

----- /cm/shared/modulefiles -----
ABINIT/8.10.1                libpng12/gcc/1.2.56
ABINIT/9.4.1-foss-2020b      libreadline/8.0-GCCcore-8.3.0
abinit/gcc/64/8.10.1         libreadline/8.0-GCCcore-9.3.0
acml/gcc-int64/64/5.3.1      libreadline/8.0-GCCcore-10.2.0
[...]
```

- Load a module (default one):

```
$ module load GCC
```

- List the loaded modules:

```
Currently Loaded Modulefiles:
1) GCCcore/10.2.0          3) binutils/2.35-GCCcore-10.2.0
2) zlib/1.2.11-GCCcore-10.2.0 4) GCC/10.2.0
```

Module commands

Modules

- Unload all loaded modules:

```
$ module purge
```

- Load more than module:

```
$ module list
Currently Loaded Modulefiles:
1) GCCcore/10.2.0          4) GCC/10.2.0
2) zlib/1.2.11-GCCcore-10.2.0 5) numactl/2.0.13-GCCcore-10.2.0
3) binutils/2.35-GCCcore-10.2.0
```

- Load GCC (version 9.3.0):

```
$ module load GCC/9.3.0
$ module list
Currently Loaded Modulefiles:
1) GCCcore/9.3.0          3) binutils/2.34-GCCcore-9.3.0
2) zlib/1.2.11-GCCcore-9.3.0 4) GCC/9.3.0
```

Remark: All loaded modules have the same GCC version.

Module commands

Install modules: EasyBuild

- List available versions:

```
$ module avail EasyBuild
----- /cm/shared/modulefiles -----
EasyBuild/4.4.2
```

- Load EasyBuild (version 4.4.2)

```
$ module list
Currently Loaded Modulefiles:
1) python/3.5.0 2) EasyBuild/4.4.2
```

- Searching for available easyconfigs files (gmsh)

```
$ eb --modules-tool EnvironmentModules --module-syntax Tcl -S gmsh
== found valid index for ↵
    /cm/shared/apps/easybuild/4.4.2/lib/python3.5/site-packages/easybuild_easyconfigs-4.4.2-py3.5.egg/easybuild/e
    so using it...
CFGS1=/cm/shared/apps/easybuild/4.4.2/lib/python3.5/site-packages/easybuild_easyconfigs-4.4.2-py3.5.egg/easybuild/e
* $CFGS1/g/gmsh/gmsh-3.0.6-foss-2017b-Python-2.7.14.eb
* $CFGS1/g/gmsh/gmsh-3.0.6-foss-2018b-Python-3.6.6.eb
* $CFGS1/g/gmsh/gmsh-4.2.2-foss-2018b-Python-3.6.6.eb
[...]
```

Module commands

Install modules: EasyBuild

- Install module:

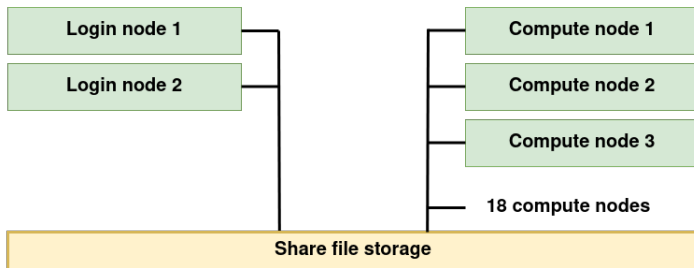
- `--robot`: to enable dependency resolution.
- `--detect-loaded-modules=error`: to print a clear error and stop when any (non-allowed) loaded modules are detected.
- `--detect-loaded-modules=purge`: to run module purge if any (non-allowed) loaded modules are detected.
- `--optarch=GENERIC`: to optimize for a generic processor architecture.

```
$ eb --modules-tool EnvironmentModules --module-syntax Tcl --prefix=/path/to/easybuild --robot ↵  
--detect-loaded-modules=error --detect-loaded-modules=purge --optarch=GENERIC ↵  
gmsh-4.7.1-foss-2020a-Python-3.8.2.eb
```

- List new modules:

```
$ module avail  
module avail gmsh-4.7.1-foss-2020a-Python-3.8.2.eb  
----- /path/to/easybuild -----  
gmsh/4.7.1-foss-2020a-Python-3.8.2
```

HPC Diagram



Login nodes are for:

- file manipulation and job script preparation
- software installation and testing
- short computational jobs (< 60 minutes and max 8 cores)
- also, be aware of the amount of memory utilized

Compute nodes are for:

- computational jobs which can use up to 44 cores
- up to 376GB memory
- jobs running > 60 minutes

Simlab Service Unit Calculations

- For the Simlab 384GB memory nodes (376GB available), you are charged Service Units (SUs) based on one of the following values whichever is greater.
 - 1 SU per CPU per hour or 1 SU per 8700MB of requested memory per hour
 - decimals are not supported by Slurm, use 8700MB instead of 8.7G

| Number of cores | Total memory per node (GB) | SU charged |
|-----------------|----------------------------|------------|
| 1 | 8.7 | 1 |
| 1 | 9 | 2 |
| 1 | 376 | 44 |
| 44 | 376 | 44 |

Nodes and Cores

- A node is one computer unit of an HPC cluster each containing memory and one or more CPUs. There are generally two classifications of nodes; login and compute. Some login and compute nodes contain GPUs.
 - login node
 - this is where users first login to stage their job scripts and do file and directory manipulations with text file editors and Unix commands.
 - compute node
 - A cluster can contain a few compute nodes or thousands of compute nodes. These are often referred to as just nodes since jobs are only scheduled on the compute nodes.
 - number of compute nodes to reserve for a job can be specified with the `-nodes` parameter
- Cores
 - Slurm refers to cores as cpus.
 - there are 40/44 cores on the Simlab
 - up 384GB memory compute nodes (376GB available)
 - number of cores can be specified with the `-cpus-per-task` parameter

Submitting Slurm Jobs

- Jobs can be submitted using a job script or directly on the command line
- A job script is a text file of Unix commands with `#SBATCH` parameters
- `#SBATCH` parameters provide resource configuration request values
 - time, memory, nodes, cpus, output files, ...
- Submit the job using `sbatch` command with the job script name
 - your job script provides a record of commands used for an analysis
 - `sbatch job_script.sh` or `job_script.slurm`
- Submit command on the command line by specifying all necessary parameters
 - must rely on your bash history to see `#SBATCH` parameters used which is not reliable
 - `sbatch -t 01:00:00 -n 1 -J myjob - -mem 7500M -o stdout.%j commands.sh`
- You can start an interactive job on the command line which ends when you exit the terminal
 - do not to use more than the requested memory and CPUs.
 - `srun - -time=04:00:00 - -mem=7G - -ntasks=1 - -cpus-per-task=1 - -pty bash`

Slurm Job Script Parameters

```
#!/bin/bash
#SBATCH --export=NONE           # do not export current env to the job
#SBATCH --partition=longq      # longq partition
#SBATCH --job-name=spades      # keep job name short with no spaces
#SBATCH --time=1-00:00:00      # request 1 day; Format: days-hours:minutes:seconds
#SBATCH --nodes=1              # request 1 node (optional since default=1)
#SBATCH --ntasks-per-node=1    # request 1 task (command) per node
#SBATCH --cpus-per-task=1      # request 1 cpu (core, thread) per task
#SBATCH --mem=7500M            # request 7.5GB total memory per node;
#SBATCH --output=stdout.%x.%j  # save stdout to a file with job name and JobID appended to file name
#SBATCH --error=stderr.%x.%j   # save stderr to a file with job name and JobID appended to file name
# unload any modules to start with a clean environment
module purge
# load software modules
module load GCC/9.3.0 Python/3.8.2-GCCcore-9.3.0
# run commands
python test.py
```

- Always include the first line exactly as it is.
- Slurm job parameters begin with `#SBATCH` and you can add comments afterwards as above
- Name the job script whatever you like but be consistent to make it easier to search for job scripts
 - `my_job_script.job`
 - `my_job_script.sh`
 - `my_job_script.slurm`

Slurm Parameters: nodes, tasks, cpus

● - -nodes

- number of nodes to use where a node is one computer unit of many in an HPC cluster (optional)
 - -nodes=1 # request 1 node (optional since default=1)
- used for multi-node jobs
 - -nodes=10
- if number of cpus per node is not specified then defaults to 1 cpu
- defaults=1 node if - -nodes not used & can use with - -ntasks-per-node and - -cpus-per-task
- do not use - -nodes with - -array

● - -ntasks

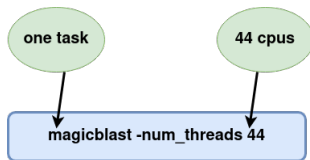
- a task can be considered a command such as blastn, bwa, script.py, etc.
- - -ntasks=1 # total tasks across all nodes per job
- when using - -ntasks without - -nodes, the values for - -ntasks-per-node and - -cpus-per-task will default to 1 node, 1 task per node and 1 cpu per task

● - -ntasks-per-node

- use together with - -cpus-per-task
- - -ntasks-per-node=1

● - -cpus-per-task

- number of CPUs (cores) for each task (command)
- - -cpus-per-task=44



Additional Slurm Parameters

- -time
 - max runtime for job (required); format: days-hours:minutes:seconds (days- is optional)
 - -time=24:00:00 # max runtime 24 hours (same as -time=1-00:00:00)
 - -time=7-00:00:00 # max runtime 7 days
- -mem
 - total memory for each node (required)
 - -mem=376G # request 376GB total memory (max available on 384gb nodes)
- -job-name
 - set the job name, keep it short and concise without spaces (optional but highly recommended)
 - -job-name=myjob
- -output
 - save all stdout to a specified file (optional but highly recommended for debugging)
 - -output=stdout.%x.%j # saves stdout to a file named stdout.jobname.JobID
- -error
 - save all stderr to a specified file (optional but highly recommended for debugging)
 - -error=stderr.%x.%j # saves stderr to a file named stderr.jobname.JobID
 - use just - -output to save stdout and stderr to the same output file: - -output=output.%j.log
- -partition
 - specify a partition (queue) to use (optional, use as needed)
 - partition is automatically assigned to short, medium, long. Also, automatic for gpu (when using -gres=gpu)

Single vs Multi-Core Jobs

- When to use single-core jobs
 - The software being used only supports commands utilizing a single-core
- When to use multi-core jobs
 - If the software supports multiple-cores (- -threads, - -cpus, ...) then configure the job script and software command options to utilize all CPUs on a compute node to get the job done faster unless the software specifically recommends a limited number of cores
 - Simlab 384GB memory compute nodes, but only 376GB can be used.

Single-Node Single-Core Job Scripts (Simlab)

● Example 1

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=8700M
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/9.3.0 Python/3.8.2-GCCcore-9.3.0

python test.py
```

● Example 2

```
#!/bin/bash
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=9g
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j

module purge
module load GCC/9.3.0 Python/3.8.2-GCCcore-9.3.0

python test.py
```

| | | |
|-----|---------------------------|---|
| 1 | Total CPUs requested | 1 |
| 1 | CPUs per node | 1 |
| 8,7 | total requested GB memory | 8 |
| 1 | SUs to start job | 2 |

Slurm Parameter: - -ntasks

```
#!/bin/bash
#SBATCH --job-name=myjob      # job name
#SBATCH --time=01:00:00      # request 1 hour
#SBATCH --ntasks=1           # request 1 task (command) per node
#SBATCH --mem=7500M          # request 7.5GB total memory per node;
#SBATCH --output=stdout.%x.%j # save stdout to a file with job name and JobID appended to file name
#SBATCH --error=stderr.%x.%j  # save stderr to a file with job name and JobID appended to file name
```

When only - -ntasks is used, the - -ntasks-per-node value will be automatically set to match - -ntasks and defaults to - -cpus-per-task=1

- - -ntasks=1
NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1
- - -ntasks=44
NumNodes=1 NumCPUs=44 NumTasks=44 CPUs/Task=1

Select GPU type on Simlab Cluster

```
#!/bin/bash
#SBATCH --job-name=my_gpu_job # job name
#SBATCH --partition=gpu       # gpu partition
#SBATCH --time=01:00:00       # request 1 hour
#SBATCH --ntasks=24           # request 24 tasks
#SBATCH --gres=gpu:1          # request 1 GPU;
#SBATCH --mem=180G            # request 180GB total memory per node;
#SBATCH --output=stdout.%x.%j # save stdout to a file with job name and JobID appended to file name
#SBATCH --error=stderr.%x.%j  # save stderr to a file with job name and JobID appended to file name

# unload modules to start with a clean environment
module purge
# load required modules
module load CUDA/11.3.1
# run your gpu command
my_gpu_command
```

There are two types of GPUs on Simlab compute nodes (1 GPU per node). To select the type of the GPU use the constraint command:

- V100: `#SBATCH - -constraint=V100` (16GB VRAM)
- P40: `#SBATCH - -constraint=P40` (22GB VRAM)

Slurm Environment Variables

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --partition=gpu
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=44
#SBATCH --mem=360G
#SBATCH --output=stdout.%x.%j
#SBATCH --error=stderr.%x.%j
module purge
module load GCC/9.3.0 Python/3.8.2-GCCcore-9.3.0

# Set the number of CPU cores to use for each task based on SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Run your parallelized application
./my_parallel_app
```

You can use the environment variable `$SLURM_CPUS_PER_TASK` to capture the value in the `#SBATCH -c` parameter so that you only need to adjust the cpus in one place

<https://slurm.schedmd.com/sbatch.html>

Slurm Parameters: `-nodes` `-ntasks-per-node`

```
#!/bin/bash
#SBATCH --export=NONE          # do not export current env to the job
#SBATCH --job-name=myjob      # job name
#SBATCH --time=1:00:00        # set the wall clock limit to 1 hour
#SBATCH --nodes=2             # request 2 nodes
#SBATCH --ntasks-per-node=1   # request 1 task (command) per node
#SBATCH --cpus-per-task=44     # request 44 cores per task
#SBATCH --mem=360G            # request 360GB of memory per node
#SBATCH --output=stdout.%x.%j # create a file for stdout
#SBATCH --error=stderr.%x.%j  # create a file for stderr
```

- `-nodes=2 -ntasks-per-node=44`
NumNodes=2 NumCPUs=88 NumTasks=88 CPUs/Task=1 mem=360G per node
- `-nodes=1 -ntasks=44`
NumNodes=1 NumCPUs=44 NumTasks=44 CPUs/Task=1 mem=360G per node
- `-nodes=2 -ntasks=88`
NumNodes=2 NumCPUs=88 NumTasks=88 CPUs/Task=1 mem=360G per node
- when `-nodes` is > 1 , make sure the software you are using supports multi-node processing
`-nodes=2 -ntasks=44`
will allocate 22 core on one node and 22 cores on a second node

Slurm Job Array Parameters and Runtime Environment Variables

- Array jobs are good to use when you have multiple samples each of which can utilize an entire compute node running software that supports multiple threads but does not support MPI
 - - `-array=0-23` # job array indexes 0-23
 - - `-array=1-24` # job array indexes 1-24
 - - `-array=1,3,5,7` # job array indexes 1,3,5,7
 - - `-array=1-7:2` # job array indexes 1 to 7 with a step size of 2
 - do not use - `-nodes` with - `-array`
- Use the index value to select which commands to run either from a text file of commands or as part of the input file name or parameter value
 - `$SLURM_ARRAY_TASK_ID` is the array index value
- stdout and stderr files can be saved per index value
 - - `-output=stdout.%x.%A.%a`
 - - `-error=stderr.%x.%A.%a`
- maximum array size is 1000 and max total pending and running jobs per user is 1000
- Limit the number of simultaneously running tasks
 - can help prevent reaching file and disk quotas due to many intermediate and temporary files
 - as one job completes another array index is run on an available node
 - - `-array=1-40%5` # job array with indexes 1-40; max of 5 running array jobs

Slurm Job Array Example 1

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --partition=special
#SBATCH --job-name=bwa_array
#SBATCH --time=00:30:00
#SBATCH --array=1-30%5      # job array of indexes 1-40; max of 5 running array indexes
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=44
#SBATCH --mem=360G
#SBATCH --output=stdout.%x.%A_%a
#SBATCH --error=stderr.%x.%A_%a
module purge
module load GCC/9.3.0
# get a line from commands.txt file into a variable named command
command=$(sed -n ${SLURM_ARRAY_TASK_ID}p commands.txt)
echo $command
```

Useful Slurm Runtime Environment Variables

- `$SLURM_CPUS_PER_TASK`
 - returns how many CPU cores were allocated on this node
 - can be used in your command to match requested `#SBATCH cpus`
 - `#SBATCH -cpus-per-task=44`
- `$SLURM_ARRAY_TASK_ID`
 - can be used to select or run one of many commands when using a job array
- `$SLURM_JOB_NAME`
 - populated by the `-job-name` parameter
 - `#SBATCH -job-name=bwa_array`
- `$SLURM_JOB_NODELIST`
 - can be used to get the list of nodes assigned at runtime
- `$SLURM_JOBID`
 - can be used to capture JobID at runtime

Useful Unix Environment Variables

- Type 'env' to see all Unix environment variables for your login session
- \$USER
 - This will be automatically populated with your NetID
echo \$USER
- \$OMP_NUM_THREADS
 - used when software uses OpenMP for multithreading; default is 1
export OMP_NUM_THREADS=44
- \$PWD
 - contains the full path of the current working directory

Submit a Slurm Job

- Submit a job
 - `sbatch my_job_file.sh`
- See status and JobID of all your submitted jobs
 - `squeue -u $USER`
- Cancel (kill) a queued or running job using JOBID
 - `scancel JOBID`
 - `scancel -u <username>`
 - `scancel -t PENDING -u <username>`
- Get an estimate of when your pending job will start
 - It is just an estimate based on all currently scheduled jobs running to the maximum specified runtime.
 - It will usually start before the estimated time.
`squeue - -start - -job JOBID`

```
[ikissami@master01 LBD]$ squeue -u $USER
```


| JOBID | PARTITION | NAME | USER | ST | TIME | NODES | ODELIST(REASON) |
|---------|-----------|-------|----------|----|------|-------|-----------------|
| 5836074 | special | myjob | ikissami | R | 0:09 | 2 | node[12-13] |


Monitor GPU usage for a Job

- nvidia-smi command

```
[ikissami@node14 LBD]$ nvidia-smi
Sun Sep 17 23:46:05 2023
```

| NVIDIA-SMI 515.65.01 Driver Version: 515.65.01 CUDA Version: 11.7 | | | | | | | | | |
|---|-----------|---------------|------------------|-------------------|----------|---------|------|------|--|
| GPU | Name | Persistence-M | Bus-Id | Disp.A | Volatile | Uncorr. | ECC | | |
| Fan | Temp | Perf | Pwr:Usage/Cap | Memory-Usage | GPU-Util | Compute | M. | | |
| 0 | Tesla P40 | On | 00000000:3B:00:0 | Off | 0 | 0 | MEM: | 0.5% | |
| MAX | 20C | P8 | 10W / 250W | 121MiB / 22.50GiB | 0% | Default | UTL: | 0% | |

[CPU:  2.8%] (Load Average: 1.01 1.04 1.05)

[MEM:  1.6%] [SWP: | 0.0%]

| Processes: ikissami@node14 | | | | | | | | | |
|----------------------------|-----|------|---------|-----|------|------|------|---------|--|
| GPU | PID | USER | GPU-MEM | %SM | %CPU | %MEM | TIME | COMMAND | |
| No running processes found | | | | | | | | | |

See Completed Job Stats

- `sacct -j JobID`
 - will only give you the following headers
JobID JobName User NCPUS NNodes State Elapsed CPUTime Start End ReqMem NodeList
- Use the following command to see MaxRSS (max memory used) and max disk write
 - `sacct - --format user,jobid,jobname,partition,nodelist,maxrss,maxdiskwrite,state,exitcode,allocres%36 -j JobID`

```
[ikissami@node14 LBD]$ sacct --format user,jobid,jobname,partition,nodelist,maxrss,maxdiskwrite,state,exitcode,allocres%36 -j 5836074
```

| User | JobID | JobName | Partition | NodeList | MaxRSS | MaxDiskWrite | State | ExitCode | AllocTRES |
|----------|---------------|---------|-----------|-------------|--------|--------------|-----------|----------|-----------------------------------|
| ikissami | 5836074 | myjob | special | node[12-13] | | | COMPLETED | 0:0 | cpu=88,mem=720G,node=2,billing=88 |
| | 5836074.batch | batch | | node12 | 2344K | 0.00M | COMPLETED | 0:0 | cpu=44,mem=360G,node=1 |

- There are two lines for this job: 1 job and 1 step
 - myjob
 - batch (JobID.batch)

sinfo command

```
HOSTNAMES FREE_MEM MEMORY AVAIL CPUS CPUS(A/I/O/T)
node01 125344 128823 up 40 16/24/0/40
node02 125373 128823 up 40 16/24/0/40
node03 309119 385483 up 44 16/28/0/44
node14 255478 385419 up 44 29/15/0/44
node15 337618 385419 up 44 36/8/0/44
node06 361152 385483 up 44 40/4/0/44
node09 262714 385419 up 44 8/36/0/44
node12 357111 385419 up 44 32/12/0/44
node13 360591 385419 up 44 32/12/0/44
...
```

- HOSTNAMES: the node name
- FREE_MEM: the free memory on the node
- MEMORY: the whole memory on the node
- CPUs: number of cpus on the node
- CPUS(A/I/O/T): A (cpus in use), I (cpus remaining)

sinfo command

```
[kissami@master01 LBD]$ sinfo -Nel
Mon Sep 18 00:50:12 2023
NODELIST      NODES PARTITION      STATE CPUS   S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
node01         1    shortq        idle   40   2:20:1 128823   51175     1 broadwel none
node01         1    defq*         idle   40   2:20:1 128823   51175     1 broadwel none
node01         1    longq         idle   40   2:20:1 128823   51175     1 broadwel none
node01         1    special        idle   40   2:20:1 128823   51175     1 broadwel none
node02         1    defq*         allocated 40   2:20:1 128823   51175     1 broadwel none
node02         1    shortq        allocated 40   2:20:1 128823   51175     1 broadwel none
node02         1    longq         allocated 40   2:20:1 128823   51175     1 broadwel none
node02         1    special        allocated 40   2:20:1 128823   51175     1 broadwel none
node03         1    defq*         mixed   44   2:22:1 385483   51175     1 skylake none
node03         1    shortq        mixed   44   2:22:1 385483   51175     1 skylake none
node03         1    longq         mixed   44   2:22:1 385483   51175     1 skylake none
node03         1    special        mixed   44   2:22:1 385483   51175     1 skylake none
node04         1    shortq        mixed   44   2:22:1 385483   51175     1 skylake none
node04         1    longq         mixed   44   2:22:1 385483   51175     1 skylake none
node04         1    defq*         mixed   44   2:22:1 385483   51175     1 skylake none
node04         1    special        mixed   44   2:22:1 385483   51175     1 skylake none
node05         1    shortq        allocated 44   2:22:1 385483   51175     1 skylake none
node05         1    defq*         allocated 44   2:22:1 385483   51175     1 skylake none
node05         1    longq         allocated 44   2:22:1 385483   51175     1 skylake none
node05         1    special        allocated 44   2:22:1 385483   51175     1 skylake none
node06         1    gpu           allocated 44   2:22:1 385483   51175     1 skylake, none
node06         1    special        allocated 44   2:22:1 385483   51175     1 skylake, none
node07         1    gpu           allocated 44   2:22:1 385483   51175     1 skylake, none
node07         1    special        allocated 44   2:22:1 385483   51175     1 skylake, none
node08         1    gpu           drained  44   2:22:1 385420   51175     1 skylake, 'Drained by cmdaemon
node08         1    special        drained  44   2:22:1 385420   51175     1 skylake, 'Drained by cmdaemon
```