Slurm Job Scheduling

Slides by Michael Dickens

Presented by Lisa M. Perez

Fall 2021

Job Scheduling (Grace)

- SBATCH Parameters
- Single node jobs
  - single-core
  - multi-core
- Multi-node jobs
  - MPI jobs
  - TAMULauncher
  - array jobs
- Monitoring job resource usage
  - at runtime
  - after job completion
  - job debugging

Slurm parameters are the same for Grace and Terra clusters except the max resources values (memory, number of cores, queue walltime) is different.

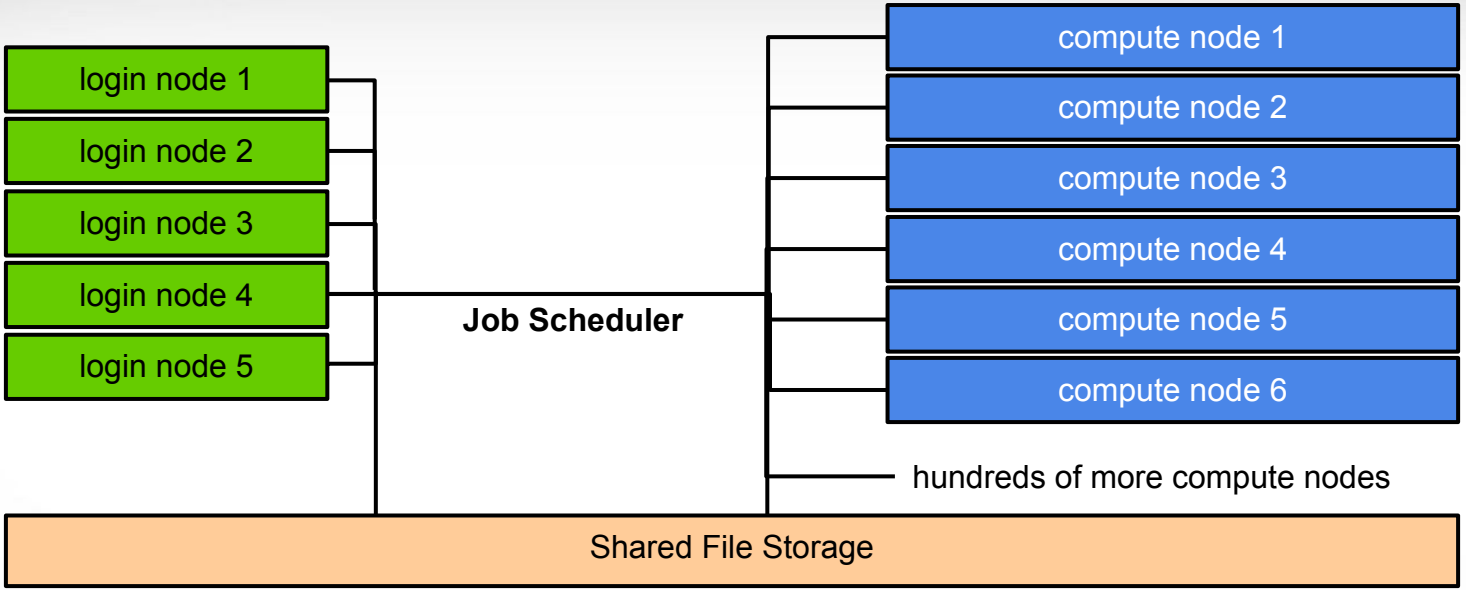
Grace Service Unit Calculations

- For the Grace 384GB memory nodes (360GB 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 7.5GB of requested memory per hour
    - decimals are not supported by Slurm, use 7500M instead of 7.5G

Number of Cores	Total Memory per node (GB)	Hours	SUs charged
1	7.5	1	1
1	8	1	2
1	360	1	48
48	360	1	48

unused SUs expire at the end of each fiscal year (Aug 31) and must be renewed

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 amount of memory utilized

compute nodes are for:

- computational jobs which can use up to 48 cores and/or up to 360GB memory (3TB for bigmem nodes) per grace compute node. jobs running > 60 minutes

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.
  - 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 48 cores on the Grace 384GB memory compute nodes (360GB avail)
  - 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`
- 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`

slurm.schedmd.com/sbatch.html

Slurm Job Script Parameters

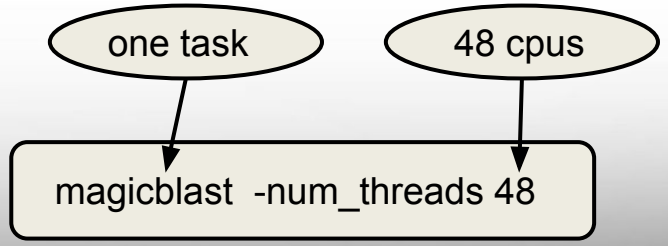
```
#!/bin/bash                    # assigns this as a bash script
#SBATCH --export=NONE          # do not export current env to the job
#SBATCH --job-name=spades      # keep job name short with no spaces
#SBATCH --time=1-00:00:00      # request 1 day; Format: days-hours:minutes:seconds
#SBATCH --nodes=1              # 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.%j     # save stdout to a file with jobID appended to name
#SBATCH --error=stderr.%j      # save stdout to a file with jobID appended to name

# unload any modules to start with a clean environment
module purge
# load software modules
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2
# run commands
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```

- Always include the first two lines exactly as they are (minus the # comments).
  - when using mpi commands with OpenMPI (foss), line two will be: --export=ALL
- 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.sbatch
  - run_program_project.sh

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 to 1 node if --nodes not used & can use together with --ntasks-per-node and --cpus-per-task
  - do not use --nodes with --array
- --ntasks

  either --ntasks, or --ntasks-per-node, or --nodes should be provided.
  - 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=48`



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=360G` `# request 360GB 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.%j` `# saves stdout to a file named stdout.JOBID`
- --error
  - save all stderr to a specified file (*optional but highly recommended for debugging*)
  - `--error=stderr.%j` `# saves stderr to a file named stderr.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)
  - only need to specify --partition parameter to use xlong, bigmem, vnc, special
  - `--partition=bigmem` `# select bigmem partion to use 3TB memory node`

Optional Slurm Parameters

- --gres
  - used to request 1 or 2 GPUs; automatically assigns --partition=gpu
  - `--gres=gpu:1` `# request 1 GPU; use :2 for two GPUs, etc`
- --account
  - specify which HPRC account to use; see your accounts with the `myproject` command
  - `--account=ACCOUNTNUMBER`
  - default account from `myproject` output is used if not specified
- --mail-user
  - send email to user
  - `--mail-user=myemail@tamu.edu`
- --mail-type
  - send email per job event: BEGIN, END, FAIL, ALL
  - `--mail-type=ALL`
- --dependency
  - schedule a job to start after a previous job successfully completes
  - `--dependency=afterok:JOBID`
    - get the JOBID of the previous job with `squeue -u $USER`

Single-Node Jobs

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
    - Grace 384GB memory compute nodes
      - 48 CPUs (cores) per compute node
      - 360GB of available memory per compute node
  - Can group multiple single-core commands into a "multi-core" job using TAMULauncher on one or multiple nodes

Single-Node Single-Core Job Scripts (Grace)

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1        # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=7500M
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 1
```

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1        # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=8G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 1
```

specify number of threads to match SBATCH parameters

Example 1

1	Total CPUs requested	1
1	CPUs per node	1
7.5	total requested GB memory	8
1	SUs to start job	2

Example 2

Slurm Parameter: --ntasks

```
#!/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 --ntasks=1                   # request 1 task (command) per node
#SBATCH --mem=7500M                  # request 7.5GB of memory per node
#SBATCH --output=stdout.%j           # create a file for stdout
#SBATCH --error=stderr.%j                 # create a file for stderr
```

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

- --ntasks=1
  - NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1
- --ntasks=48
  - NumNodes=1 NumCPUs=48 NumTasks=48 CPUs/Task=1

Requesting all CPUs and Available Memory on Grace Compute Nodes

48 cores, 384 GB nodes (800 nodes available)

```
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
```

48 cores, 360 GB memory A100 **GPU** nodes (100 nodes available)

```
#SBATCH --gres=gpu:a100:2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
```

48 cores, 384 GB T4 **GPU** nodes (8 nodes available)

```
#SBATCH --gres=gpu:t4:4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
```

80 cores, 3 TB memory nodes (8 nodes available)

```
#SBATCH --partition=bigmem
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=80
#SBATCH --mem=2929G
```

Requesting one GPU on Grace Compute Nodes

Request ½ the available resources on A100 and RTX 6000 GPU nodes when using 1 GPU so that someone else can use the other GPU. Request ¼ the resources for T4 GPU so someone else can use the other 3 GPUs.

```
#SBATCH --gres=gpu:a100:1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=24
#SBATCH --mem=180G
```

```
#SBATCH --gres=gpu:rtx:1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=24
#SBATCH --mem=180G
```

```
#SBATCH --gres=gpu:t4:1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --mem=90G
```

Select GPU type on Grace Cluster

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=my_gpu_job
#SBATCH --time=1-00:00:00
#SBATCH --ntasks=24
#SBATCH --gres=gpu:a100:1          # request 1 A100 GPU; use :2 for two GPUs
#SBATCH --mem=180G                 # can request max of 360GB on GPU nodes
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

# 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 three types of GPUs on Grace compute nodes. Select the type and quantity with --gres
  - A100: --gres=gpu:a100:N (N can be 1 or 2) (100 total A100 compute nodes available)
  - RTX6000: --gres=gpu:rtx:N (N can be 1 or 2) (9 total RTX6000 compute nodes available)
  - T4: --gres=gpu:t4:N (N can be 1, 2, 3, or 4) (8 total T4 compute nodes available)

Single-Node Multi-Core Job Scripts

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1     # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=24
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 14
```

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1     # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 48
```

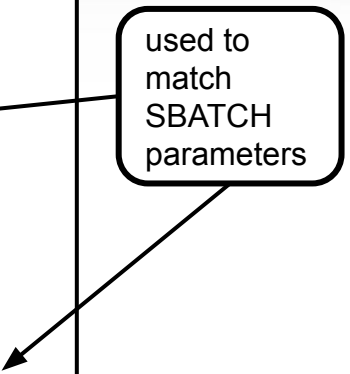
specify number of threads to match SBATCH parameters

24	Total CPUs requested	48
24	CPUs per node	48
360	total requested GB memory	360
1152	SUs to start job	1152

It is best to request all cores and all memory if using the entire node

Slurm Environment Variables

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads $SLURM_CPUS_PER_TASK
```



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

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

Multi-Node Jobs

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=48           # request 48 cores per task
#SBATCH --mem=360G                   # request 360GB of memory per node
#SBATCH --output=stdout.%j           # create a file for stdout
#SBATCH --error=stderr.%j            # create a file for stderr
```

It is easier to scale jobs by using --nodes with --ntasks-per-node instead of with --ntasks.
If you use --nodes with --ntasks, you need to calculate total CPUs for all nodes as the --ntasks value

- --nodes=2 --ntasks-per-node=48
  - NumNodes=2 NumCPUs=96 NumTasks=96 CPUs/Task=1 mem=360G per node
- --nodes=1 --ntasks=48
  - NumNodes=1 NumCPUs=48 NumTasks=48 CPUs/Task=1 mem=360G per node
- --nodes=2 --ntasks=96
  - NumNodes=2 NumCPUs=96 NumTasks=96 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=48
    - will allocate 24 core on one node and 24 cores on a second node

MPI Multi-Node Multi-Core Job Script: Example 1

```
#!/bin/bash
#SBATCH --export=NONE    # use ALL for non-Intel MPI (foss) and NONE for Intel MPI
#SBATCH --job-name=moose
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load MOOSE/20171003-intel-2017A-Python-2.7.12-CUDA-8.0.44
mpirun -np 480 -npernode 48 /path/to/moose-opt -i moose.i
```

480	Total CPUs requested
48	CPUs per node
360	total requested GB memory per node
11,520	SUs to start job (nodes * cpus * hours)

MPI Multi-Node Multi-Core Job Script: Example 2

```
#!/bin/bash
#SBATCH --export=ALL      # use ALL for non-Intel MPI (foss) and NONE for Intel MPI
#SBATCH --job-name=abyss-pe
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/8.3.0  OpenMPI/3.1.4 ABySS/2.1.5
abyss-pe np=480 j=48 lib='lib1' lib1='sample1_R1 sample1_R2'
```

480	Total CPUs requested
48	CPUs per node
360	total requested GB memory per node
11,520	SUs to start job (nodes * cpus * hours)

TAMULauncher

- hprc.tamu.edu/wiki/SW:tamulauncher
- Use when you have hundreds or thousands of commands to run each utilizing a single-core or a few cores
  - tamulauncher keeps track of which commands completed successfully
    - to see the list of completed commands
      - `tamulauncher --status commands_file.txt`
    - if time runs out, then tamulauncher can be restarted and it will know which was the last successfully completed command
  - submit tamulauncher as a batch job within your job script
  - can run tamulauncher interactively on login node; limited to 8 cores
  - you can check the --status on the command line from the working directory
- run a single command of your thousands to make sure the command is correct and to get an estimate of resource usage (CPUs, memory, time)
- request all cores and memory on the compute node(s) and configure your commands to use all available cores

TAMULauncher Multi-Node Single-Core Commands

commands.txt (500 lines for example)

```
spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 1
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 1
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 1
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 1
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 1
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 1
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 1
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 1
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 1
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 1
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 1
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 1
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 1
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 1
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 1
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 1
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 1
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 1
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 1
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 1
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 1
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```

run_spades_tamulauncher.sh

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=48
#SBATCH --cpus-per-task=1
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2

tamulauncher commands.txt
```

run 48 spades.py commands per node with each command using 1 core.
Requesting all 48 cores on Grace reserves entire node for your job

- run 48 single-core commands per node; useful when each command requires < 7.5GB memory
- create a commands file (named whatever you want) to go with the the job script
- load the software module in the job script not the commands file

TAMULauncher Multi-Node Multi-Core Commands

commands.txt (500 lines for example)

```
spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 4
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 4
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 4
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 4
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 4
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 4
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 4
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 4
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 4
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 4
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 4
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 4
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 4
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 4
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 4
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 4
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 4
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 4
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 4
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 4
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 4
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 4
```

run_spades_tamulauncher.sh

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=4
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module purge
module load GCC/9.3.0  SPAdes/3.14.1-Python-3.8.2

tamulauncher commands.txt
```

run 12 spades.py commands per node with each command using 4 cores. Requesting all 48 cores on Grace reserves entire node for your job

- useful when each command requires more than 7.5GB but less than all available memory
- use OMP_NUM_THREADS if needed when running fewer commands than requested cores
  - add on the line before the tamulauncher command
  - `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`

Making a TAMULauncher Commands File Part 1

Input files are two files per sample and named: s1_R1.fastq.gz s1_R2.fastq.gz
Run this command to create the example files:

```
mkdir seqs && touch seqs/s{1..40}_R{1,2}.fastq.gz
```

Run the following commands to get familiar with useful shell commands for creating and manipulating variables

```
file=seqs/s1_R1.fastq.gz
echo $file
basename $file
sample=$(basename $file)
echo $sample
echo ${sample/_R1.fastq.gz}
echo ${sample/R1/R2}
```

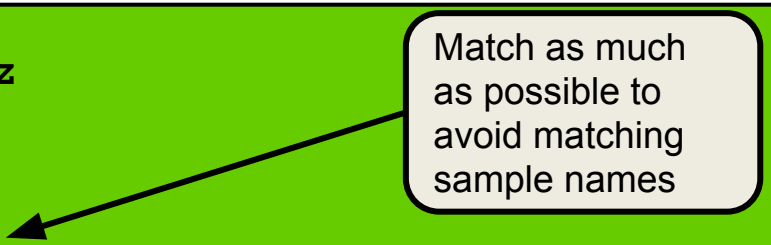
```
# create variable named file
# show contents of variable
# strip off path of variable
# create variable named sample
# show contents of variable
# strip off _R1.fastq.gz
# substitute text R1 with R2
```

Making a TAMULauncher Commands File Part 2

Input files are two files per sample and named: s1_R1.fastq.gz s1_R2.fastq.gz

Run the following commands to loop through all R1 files in the reads directory and create the commands.txt
Use just the R1 files because we only need to capture the sample names once.

```
for file in seqs/*_R1.*gz
do
read1=$file
read2=${read1/_R1./_R2.}
sample=$(basename ${read1/_R1.fastq.gz})
echo spades.py -1 $read1 -2 $read2 -o ${sample}_out --threads 1
done > commands.txt
```



Other Useful Unix Commands

```
${variable#*SubStr}      # will drop beginning of variable value up to first occurrence of 'SubStr'
${variable##*SubStr}     # will drop beginning of variable value up to last occurrence of 'SubStr'
${variable%SubStr*}      # will drop part of variable value from last occurrence of 'SubStr' to the end
${variable%%SubStr*}     # will drop part of variable value from first occurrence of 'SubStr' to the end
```

These are useful if the part of the filename for each sample that needs to be removed is not the same.

s1_S1_R1.fastq.gz **s2**_S2_R1.fastq.gz **s3**_S3_R1.fastq.gz

want to remove this part from each file name

Make a new directory and create a new set of files for this exercise.

```
mkdir seqs2
for i in {1..10}; do touch seqs2/s${i}_S${i}_R{1,2}.fastq.gz; done
```

Unix Command	Output
`file=seqs2/s1_S1_R1.fastq.gz`	
`echo $file`	`seqs2/s1_S1_R1.fastq.gz`
`echo ${file%_S*}`	`seqs2/s1`
`basename ${file%_S*}`	`s1`
`sample=$(basename ${file%_S*})`	
`echo $sample`	`s1`

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-%A_%a`
  - `--error=stderr-%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 --job-name=bwa_array
#SBATCH --time=1-00:00:00
#SBATCH --array=1-40%5      # job array of indexes 1-40; max of 5 running array indexes
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
#SBATCH --output=stdout.%A_%a
#SBATCH --error=stderr.%A_%a

module purge
module load GCC/9.3.0  BWA/0.7.17
# get a line from commands.txt file into a variable named command
command=$(sed -n ${SLURM_ARRAY_TASK_ID}p commands.txt)
$command
```

- The sed command will print a specified line number from commands.txt based on the **$SLURM_ARRAY_TASK_ID**
- The number of lines in your commands.txt file should be the same as the number of array indexes
- Can use %5 to limit the array to a maximum of 5 nodes used simultaneously but you need enough SUs to cover entire number of array indexes in order to submit the job. May be useful to prevent creating too many temporary files.
- There are other ways to use $SLURM_ARRAY_TASK_ID but this example is useful because it has a file of all commands used in each array index
- Doesn't work when commands have redirection operators: | < >

Slurm Job Array Example 2

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=bwa_array
#SBATCH --time=1-00:00:00
#SBATCH --array=1-40          # run all 40 array indexes simultaneously using 40 nodes
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --mem=360G
#SBATCH --output=stdout.%A_%a
#SBATCH --error=stderr.%A_%a

module purge
module load GCC/9.3.0  BWA/0.7.17

bwa mem -M -t 48 -R '@RG\tID:\tLB:pe\tSM:DR34\tPL:ILLUMINA' genome.fasta \
sample_${SLURM_ARRAY_TASK_ID}_R1.fastq.gz  sample_${SLURM_ARRAY_TASK_ID}_R2.fastq.gz \
| samtools view -h -Sb - | samtools sort -o sample_${SLURM_ARRAY_TASK_ID}.out.bam \
-m 7G -@ 1 -T $TMPDIR/tmp4sort${SLURM_ARRAY_TASK_ID} -
```

- Can use **$SLURM_ARRAY_TASK_ID** if your commands only differ by a number in the file names
- The **$SLURM_ARRAY_TASK_ID** variable will be assigned the array index from 1 to 40 in this example
- Can use this approach when commands have redirection operators: | < >

Useful Slurm Runtime Environment Variables

- $TMPDIR
  - this is a temporary local disk space (~1.4TB) created at runtime and is deleted when the job completes
  - the directory is mounted on the compute node and files created in $TMPDIR do not count against your file and disk quotas
  - `samtools sort -T $TMPDIR/sort`
- $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=48`
    - `samtools sort --threads $SLURM_CPUS_PER_TASK`
- $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`**
- $SCRATCH
  - You can use this to change to your /scratch/user/netid directory
    - **`cd $SCRATCH`**
- $OMP_NUM_THREADS
  - used when software uses OpenMP for multithreading; default is 1
    - **`export OMP_NUM_THREADS=48`**
- $PWD
  - contains the full path of the current working directory

Monitoring Job Resource Usage

Submit a Slurm Job

- Submit a job
  - **`sbatch my_job_file.sh`**
- See status and JOBID of all your submitted jobs
  - **`squeue -u $USER`**

```
Thu Oct 17 11:32:44 2019
JOBID     PARTITION  NAME     USER      STATE     TIME      TIME_LIMIT  NODES NODELIST(REASON)
3279434   short,med  bwa      netid     PENDING       0:00   1-00:00:00    1 (Priority)
```

- Cancel (kill) a queued or running job using JOBID
  - **`scancel JOBID`**
- 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`**

Monitor CPU usage for a Grace Running Job

- See status and JOBID of all your submitted jobs
  - `squeue -u $USER`
- See CPU and memory usage of all your running jobs
  - `pestat -u $USER`
  - stats for pestat are updated every **3** minutes
  - can use with watch command to run pestat every 2 seconds; updates are still every 3 minutes
    - `watch pestat -u $USER`

```
Hostname   Partition  Node    Num_CPU  CPUload  Memsize  Freemem  Joblist
                      State   Use/Tot           (MB)     (MB)     JobId User ...
c447         long*    alloc   48   48   46.56*   368640   359957   731601 netid
c466         long*    alloc   48   48   46.78*   368640   360747   731601 netid
c499         long*    alloc   48   48   45.65*   368640   361448   731601 netid
c514         long*    alloc   48   48   47.10*   368640   361524   731601 netid
c521         long*    alloc   48   48   48.01*   368640   361277   731601 netid
```

Low CPU load utilization highlighted in **Red**
Good CPU load utilization highlighted in **Purple**
Ideal CPU load utilization displayed in White
(Freemem should also be noted)

Monitor GPU usage for a Running Job

- Leave 1 CPU and 1GB memory from the max available resources for the first job in order to launch a second job on the same compute node that will be used to monitor the first job
- See the NODELIST in output of
  - **`squeue -u $USER`**

```
JOBID     NAME    USER     PARTITION   NODES  CPUS  STATE      TIME    TIME_LEFT    START_TIME           REASON  NODELIST
3276433   bwa     netid    short       1      27    RUNNING    5:33    54:27        2019-10-15T10:52:06  None    c127
```

- Launch a second job to run on the same node as your first job
  - **`srun --nodelist c127 --time 01:00:00 --mem 1G --pty bash`**
- type in the hostname command to see that you are logged into the compute node
  - hostname
    - **`c127.cluster`**
- run the nvidia-smi command in addition to the Unix watch command
  - **`watch nvidia-smi`**
  - use **`Ctrl+c`** to exit the watch command
- you can also monitor real time CPU usage with the top command
  - **`top -u $USER`**

https://hprc.tamu.edu/wiki/Bioinformatics:FAQ#Monitor_resource_usage_for_a_running_job

Monitor a Running Job

- See lots of info about your running or recently completed (~10 minutes) job
  - **`scontrol show job JOBID`**
- gpu jobs are charged at a rate of 24x3 SUs per A100 or RTX 6000 gpu and 12X3 per T4 requested on Grace
- can add this command at the end of your job script to capture job info into the stdout file
  - **`scontrol show job $SLURM_JOBID`**

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=abyss
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=48
#SBATCH --cpus-per-task=1
#SBATCH --mem=360G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j
```

```
JobId=836933 JobName=abyss
   UserId=mynetid(14499) GroupId=mynetid(14499) MCS_label=N/A
   Priority=78034 Nice=0 Account=132787216288 QOS=normal
   JobState=PENDING Reason=Priority Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=1-00:00:00 TimeMin=N/A
   SubmitTime=2021-09-09T13:08:02 EligibleTime=2021-09-09T13:08:02
   AccrueTime=2021-09-09T13:08:02
   StartTime=2021-09-09T16:46:20 EndTime=2021-09-10T16:46:20 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2021-09-09T13:08:48
   Partition=short,medium,long AllocNode:Sid=login2:80769
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=(null) SchedNodeList=c571
   NumNodes=1-1 NumCPUs=48 NumTasks=48 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=48,mem=360G,node=1,billing=48
   Socks/Node=* NtasksPerN:B:S:C=48:0:*:* CoreSpec=*
   MinCPUsNode=48 MinMemoryNode=360G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/scratch/user/mynetid/myjobdir/run_abyss_1.9.0_pe_grace.sh
   WorkDir=/scratch/user/mynetid/myjobdir
   Comment=(from job_submit) your job is charged as below
            Project Account: 132878216828
            Account Balance: 199883.903045
            Requested SUs:   1152
   StdErr=/scratch/user/mynetid/myjobdir/stderr.836933
   StdIn=/dev/null
   StdOut=/scratch/user/mynetid/myjobdir/stdout.836933
   Power=
   NtasksPerTRES:0
```

See Completed Job Efficiency Stats

- **`seff JOBID`**
  - will show CPU and Memory efficiency based on selected resources

```
Job ID: 836933
Cluster: grace
User/Group: mynetid/mynetid
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 48
CPU Utilized: 00:29:36
CPU Efficiency: 26.81% of 01:50:24
core-walltime
Job Wall-clock time: 00:02:18
Memory Utilized: 40.60 GB
Memory Efficiency: 11.28% of 360.00 GB
```

CPU load was at 100% for 27% of the run time

max memory utilized was 11% of requested memory

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,alloctres%36 -j JOBID
```

```
User      JobIDJobName   Partition    NodeList       MaxRSS      MaxDiskWrite State     ExitCode            AllocTRES
--------- ------------ ---------- ---------- -------------- ---------- ----------- ---------- -------- ------------------------------------
mynetid   836933        abyss     medium        c432                                COMPLETED    0:0     billing=48,cpu=48,mem=360G,node=1
          836933.batch  batch                   c432        42569196K     964.21M   COMPLETED    0:0                 cpu=48,mem=360G,node=1
          836933.exte+  extern                  c432             872K           0   COMPLETED    0:0     billing=48,cpu=48,mem=360G,node=1
```

926 MB max writing to disk

- There are three lines for this job: 1 job and 2 steps
  - `spades_pe`
  - `batch (JobID.batch)`
  - `extern (JobID.extern)`

you can set an alias in your .bashrc for the sacct --format command

See All Your Jobs for Current Fiscal Year

- `myproject -j all`
  - ProjectAccount
  - JobID
  - JobArrayIndex
  - SubmitTime
  - StartTime
  - EndTime
  - Walltime
  - TotalSlots
  - UsedSUs
  - Total Jobs
  - Total Usage (SUs)

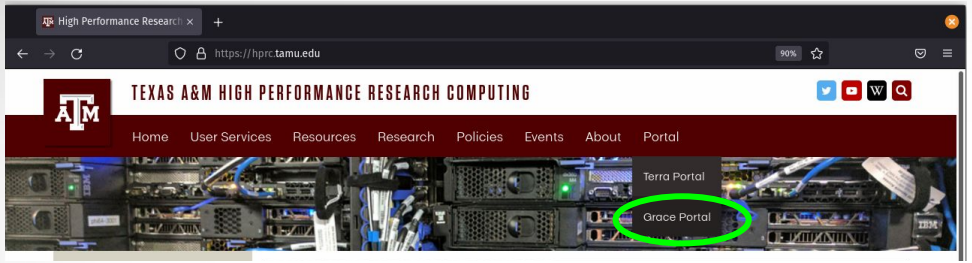
Debugging Slurm Jobs

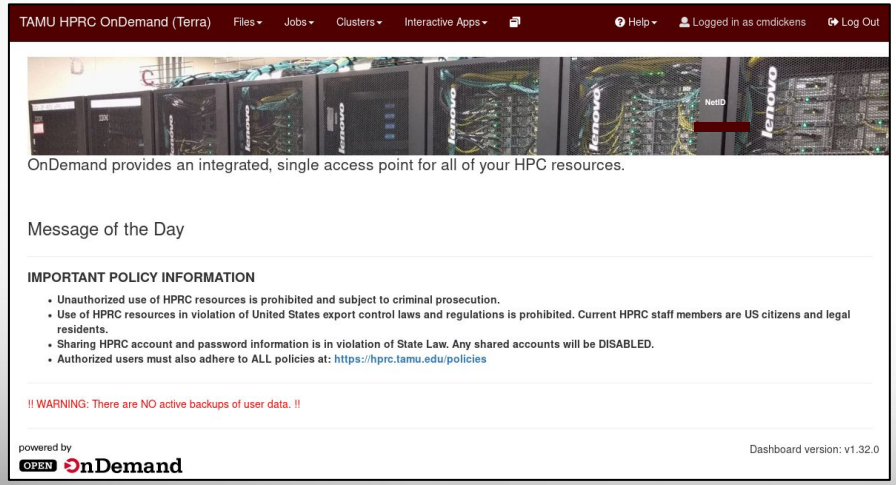
- If job was not scheduled, check your HPRC account to see if you have enough SUs
  - **`myproject`**
- Look for an out of memory error message; could occur in only one index of a job array

```
slurmstepd: error: Exceeded job memory limit at some point.
```

  - Increase the amount of SBATCH memory in your job script and resubmit the job
- If you see an '*Out of disk space*' or '*No space left on device*' error
  - check your file and disk quotas using the showquota command
    - **`showquota`**
  - reduce the number of files you have generated
    - delete any nonessential or temporary files
    - use **`$TMPDIR`** in your command if software supports a temporary directory
    - create a .tar.gz package of completed projects to free up disk space
  - request an increase in file and/or disk quota for your project

portal.hprc.tamu.edu





The HPRC portal allows users to do the following

- Browse files on the the Grace filesystem
- Access the Grace Unix command line
  - no SUs charged for using command line
  - runs on login node; limit your processes to 8 cores
- Launch jobs
  - SUs charged when launching jobs
- Compose job scripts
- Launch interactive GUI apps (SUs charged)
- Monitor and stop running jobs and interactive sessions
- Grace
  - ANSYS Workbench
  - Abaqus/CAE
  - MATLAB
  - VNC
  - Jupyter Notebook, JupyterLab
  - BEAUti
  - Gap5
  - IGV
  - Mauve
  - Structure
  - RStudio

For More HPRC Help...

Website: hprc.tamu.edu
Email: help@hprc.tamu.edu
Telephone: (979) 845-0219
Visit us in person: Henderson Hall, Room 114A
Blocker 217
(best to email or call in advance and make an appointment)

Help us, help you -- we need more info

- Which Cluster
- UserID/NetID
- Job id(s) if any
- Location of your jobfile, input/output files
- Application used if any
- Module(s) loaded if any
- Error messages
- Steps you have taken, so we can reproduce the problem

Let us know when the issue has been resolved so we can close the helpdesk ticket