Slurm

Jump To:

- **Command Equivalents (Slurm<->LSF)**
- **Slurm Documentation**
- **Examples**

Attention:

As of Dec. 17, 2014 the new Tufts cluster uses **slurm** for job scheduling. User job types such as serial single processor, MPI multiprocessor, GUI-based interactive applications, SMP threaded applications and batch jobs may be run on compute nodes using **slurm**. This means that there is no LSF bsub command on the cluster to submit jobs to queues as before. Also there are no queues. Note, slurm tastes different.

Cluster Assistance

For additional information, please contact via email Research Technology Services at tts-research@tufts.edu

Overview of slurm Concepts

Slurm is an open-source workload manager designed for Linux clusters of all sizes. It provides three key functions. First it allocates **exclusive** and/or **non-exclusive** access to resources (computer nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work on a set of allocated nodes. Finally, it arbitrates contention for resources by managing an internal queue of pending work. This is done with slurm's fair-share algorithm framework, similar to LSF, but we are not using job preemption or suspension.

Partitions vs. queues

LSF has the concept of queues. LSF queues on the old cluster differentiates service and job types. **Slurm** does not have queues and instead has the concept of a **partition**. The new cluster, **login.cluster.tufts.edu**, has five partitions: **batch, interactive, gpu, largemem and mpi**. When a compute job is submitted with **slurm**, it must be placed on a **partition**. The **batch** partition is the default partition. Each partition has default settings of various kinds, such as cores, memory, duration, priority, threads, etc... Explicit slurm requests is required to alter defaults. For example, the batch partition has a default allocation of one cpu core and a maximum of 2gig. of ram memory.   This could be used for a serial job or shell script. Partition settings may be viewed using the **squeue** command.

salloc vs. bsub options

Another concept is that of resource allocation. The **salloc** command allows one to request resources and a partition. **salloc** is available to fine tune resources for one or more submitted jobs. Cavalier use of **salloc** will likely waste resources unless it is well thought out. Try to assess your needs carefully and not over allocate. It is likely that the bigger the requested allocation, the longer the wait for the resources to be granted.   Associated with an allocation via the **salloc** command is the notion of **job steps**. These **steps** are nested within a granted allocation.   An allocation of resources is relinquished when you are finish with your job steps(by typing **exit** when used interactively) or when the allocation times out or tasks are exhausted or the use of **exit** in a script is executed. **Note, for most simple and straight forward slurm needs, most users will not need to use salloc.** Commands **sbatch** and **srun** can host all the typical resource requests for most jobs.

slurm Qos vs. queues

The old cluster used several LSF queues to provide a quality of service distinction by mechanisms such as relative job priority, hardware separation, fairshare policy rules and preemption. This approach supported the needs of faculty contributed hardware owners. An equivalent solution in slurm is the concept of **slurm bank account and qos**. This is optionally applied to a partition and job submission.   Most cluster users are not affected by this and can ignore it, resulting in the profile of **normal qos**. Nothing special needs to be done by cluster users. The normal qos is specified as an option to **srun** and **sbatch** commands as follows:

--account=normal   --qos=normal

Those users that are affected, and require access as define by previous **LSF** queues may use the following as appropriate:

```
--account=yushan   --qos=yushan

--account=abriola   --qos=abriola

--account=napier   --qos=napier

--account=miller   --qos=miller

--account=cowen   --qos=cowen

--account=khardon   --qos=khardon

--account=cbi  --qos=cbi

--account=atlas  --qos=atlas

--account=qu  --qos=qu
```

NOTE:

These **account** and **qos** pairs have elevated slurm placement privileges.

Finally, an additional special purpose QoS, **dregs**, is provided to address low priority long running jobs that can withstand pre-emption and cancellation as a possible outcome. **This is only suitable for a special class of jobs.**

--account=dregs --qos=dregs

Note: dregs is configured to have the lowest job priority among all slurm qos/account options. Jobs submitted with a dregs specification may be preempted by other jobs during periods of heavy cluster use. If slurm decides to preempt dregs job(s), the job(s) are re-queued for submission and subject to available cluster resources. In effect the job starts over! Not all work loads are appropriate for this qos treatment. Single core/low memory jobs are more likely to be suitable for dregs access and successful use.

Partition runtime duration limit:

Partition	duration
gpu	3 days
largemem	5 days
interactive	4 hours
batch	3 days
mpi	7 days
m4	7 days

IMPORTANT: Your job duration estimate

Use of slurm requires a job duration estimate. A default job time limit of 15 minutes has been set since May 26, 2016. Do not confuse this setting with a slurm partition maximum duration time limit, which already exists. Each submitted job to all partitions(except Interactive), needs to have a time estimate of your job's duration. For example, if your job is expected to run for 1 day and 2 hours, specify that in the **sbatch** or **srun** options:

```
> sbatch .... -p batch --time=1-2:00:00 .....
```

NOTE: Failure to not include the **--time** specification will result in the job's termination after the default time limit of 15 minutes(plus some overhead time). And if your job exceeds your time estimate, slurm will then kill the job. It is important for you to figure this out. This will require some experimentation on your part. Experienced users will have some knowledge of this. However, new job initial conditions may not be well understood. If you think your job may be a long duration job, set the **--time** option to something close to the partition maximum time and note the resulting duration. Subsequent job submissions may then be specified with better accuracy. Note, the exact time is not needed!

Note: The gpu partition contains model K20 Nvidia gpus on Cisco nodes. The following nodes have older gpus, but cuda support is not enabled.

M3 nodes with NVIDIA GPUs are m3n45 and m3n46
M4 nodes with NVIDIA GPUs are m4c29 and m4c60

Unless there is a good reason to use these, we suggest using the **gpu partition.**

Compute nodes:

The new cluster compute nodes follow a different naming scheme from the old cluster. This allows and simplifies cluster support logistics. It is good practice to not hard-code any compute node names. Compute node redundancy and support logistics benefit from node name abstraction. This allows slurm to place jobs independent of node names.

Some command equivalents:

	slurm	**LSF**
kill a job	scancel	bkill
what are my running job(s)	squeue	bjobs, qstat
submit a job	sbatch	bsub
what queues/partitions	squeue	bqueues
how to obtain resources	salloc	bsub + specific queue + bsub options
node listing	sinfo	bhosts, lshosts
interactive session or application	srun	bsub -Ip -q int_public6 xxxxxx
controlling jobs	scontrol	bstop
usage accounting	sreport, sacct, sstat	bacct
other submissions	srun	lsrun, lsplace, bsub
copy files to allocated local disk	sbcast	lsrcp

Additional detail for some command equivalents:

User Commands	**Slurm**	**LSF**
Job submission	sbatch [script_file]	bsub [script_file]
Job deletion	scancel [job_id]	bkill [job_id]
Job status (by job)	squeue [job_id]	bjobs [job_id]
Job status (by user)	squeue -u [user_name]	bjobs -u [user_name]
Job hold	scontrol hold [job_id]	bstop [job_id]
Job release	scontrol release [job_id]	bresume [job id]
Queue list	squeue	bqueues
Node list	sinfo -N OR scontrol show nodes	bhosts
Cluster status	sinfo	bqueues
GUI	sview	xlsf OR xlsbatch
Environment	**Slurm**	LSF
Job ID	$SLURM_JOBID	$LSB_JOBID

Submit Directory	$SLURM_SUBMIT_DIR	$LSB_SUBCWD
Submit Host	$SLURM_SUBMIT_HOST	$LSB_SUB_HOST
Node List	$SLURM_JOB_NODELIST	$LSB_HOSTS/$LSB_MCPU_HOST
Job Array Index	$SLURM_ARRAY_TASK_ID	$LSB_JOBINDEX
Job Specification	**Slurm**	**LSF**
Script directive	#SBATCH	#BSUB
Queue	-p [queue]	-q [queue]
Node Count	-N [min[-max]]	-n [count]
CPU Count	-n [count]	-n [count]
Wall Clock Limit	-t [min] OR -t [days-hh:mm:ss]	-W [hh:mm:ss]
Standard Output File	-o [file_name]	-o [file_name]
Standard Error File	e [file_name]	-e [file_name]
Combine stdout/err	(use -o without -e)	(use -o without -e)
Copy Environment	--export=[ALL | NONE | variables]	
Event Notification	--mail-type=[events]	-B or -N
Email Address	--mail-user=[address]	-u [address]
Job Name	--job-name=[name]	-J [name]
Job Restart	--requeue OR --no-requeue (NOTE:configurable default)	-r
Working Directory	--workdir=[dir_name]	(submission directory)
Resource Sharing	--exclusive OR --shared	-x
Memory Size	--mem=[mem][M|G|T] OR --mem-per-cpu=[mem][M|G|T]	-M [MB]
Account to charge	--account=[account]	-P [account]
Tasks Per Node	--tasks-per-node=[count]	
CPUs Per Task	--cpus-per-task=[count]	
Job Dependency	--depend=[state:job_id]	-w [done | exit | finish]
Job Project	--wckey=[name]	-P [name]
Job host preference	--nodelist=[nodes] AND/OR --exclude=[nodes]	-m [nodes]
Quality Of Service	--qos=[name]	
Job Arrays	--array=[array_spec](Slurm version 2.6+)	J "name[array_spec]"
Generic Resources	--gres=[resource_spec]	
Begin Time	--begin=YYYY-MM-DD[THH:MM[:SS]]	-b[[year:][month:]day:]hour:minute

Getting Started

Slurm documentation section and examples can be found on this page. If you are just migrating to slurm, a one page quick overview orientation is available, Q uickStartSlurm. Otherwise please view recent introductory slurm Brown Bag Powerpoint session files below.

slurm Documentation

- QuickStartSlurm overview
- slurm documentation is available on the **slurm** site.
- Useful slurm **FAQs**
- Man pages are available as well on the login node.
  > man -k slurm
- A comparison matrix of slurm, lsf and other popular scheduler commands can be found **here**
- Online tutorials can be found at: http://slurm.schedmd.com/tutorials.html
- slurm mpi docs: mpi_guide
- Introduction To Slurm Brown Bag Powerpoint session file (PDF)

FairShare scheduling

Slurm scheduling on the cluster will place jobs according to its FairShare algorithm which avoids queuing by first come first serve. All users have a maximum access limit of 412 cores/cpus (if available). There is no limit on the number of jobs submitted. The cluster has about 4000+ cores/cpus. About 3000 of them are often in use and it can be busy from day to day.

FairShare considers job requests for resources and compares to available resources. It then tries to align resources with requests while considering a user's slurm history. Depending on the mix of resource requests, available resources, user "hog" history, and other constraints at the time, slurm will place jobs fairly. It has proven to work very well and is why it is used at almost all HPC compute centers. Slurm is aware of large numbers of PENDING jobs and this is in the fairshare calculation for placement. We suggest that you submit your jobs so as to "get in line" and not miss next opportunities for placement.

Note, the batch partition has a limit of 3 days. If your jobs need to run longer consider the largemem, mpi or dregs partitions. It is important for you to understand your job's resource needs. This will help to avoid slurm killing your jobs when default resources are exceeded. slurm strictly enforces it's constraints!

Modules

Modules are required on the cluster. These are needed to access an application and it's software environment. Since there are often several versions of a particular software item, always check what is available. The default module for a particular item may change. To view what is available:

```
> module available
```

Note, **please do not** run application software on the login/headnode of the cluster. Instances will be **terminated** and a reminder sent to you. Submit compute jobs to compute nodes with slurm.

Overview of typical use case:

- The **salloc** command will address your partition and hardware environment requests.
- Next, modules corresponding to software of interest should be invoked on the command line or within batch scripts.
- Next step, submit a job to the reserved allocation.

slurm Examples:

How to view slurm partitions and additional info:

> sinfo

For additional info:

```
> sinfo -Nel

> sinfo  -p  batch

> sinfo -p mpi
```

What jobs do I have running and provide information on the following resources requested:

```
> squeue  -u your_utln

or for more details that you can customize sacct arguments:

>  sacct -u your_utln  --format=jobid,elapsed,ncpus,ntasks,state,NNodes,partition,MaxVMSize,MaxRSSNode
```

I have a lot of jobs pending, when will they start?

```
> squeue --start --job  your_job_id

And how busy is the cluster?


> sreport cluster utilization
```

The scontrol command is useful for many tasks. Note, there is a difference between slurm Running and Pending jobs when it comes to controlling their state.

```
For pending jobs pause/resume use scontrol options: "hold"/"release"
For running jobs pause/resume use scontrol options:  "suspend"/"resume"
```

How may I obtain information about my job so that I can verify what I requested is indeed granted?

```
Suppose the job ID is 123456789  use command:

> scontrol show job 123456789
```

Is there a graphical client to view job states instead of command line tools?
> sview

How to cancel an allocation:

Create a request for one node with one cpu.

```
> salloc -N1 -n1

salloc: Granted job allocation 322

> squeue

      JOBID PARTITION  NAME     USER      ST     TIME  NODES NODELIST(REASON)

        322    batch        bash       your_utln  R      0:05     1          alpha021

> scancel 322

salloc: Job allocation 322 has been revoked.
```

Or type:

```
> exit
```

Other scancel options for many jobs:

```
> scancel --state=PENDING --user=your_utln  --partition=batch

> scancel --state=RUNNING  --user=your_utln  --partition=batch
```

How to obtain a one job step xterm session in the default batch partition for 2 hours of work:

```
> srun --x11=first --pty  --time=0-2:00:00  xterm
```

or in the interactive partition:

```
> srun --x11=first --pty -p interactive xterm
```

How to send a simple command to a node:

```
>  srun --pty -p interactive hostname

alpha021

>
```

How to allocate 1 compute node with 1 core and with an xterm window accessing the batch partition for 2 hours:

```
> salloc -N1 -p batch

> srun   --pty    --x11=first --time=0-2:00:00  xterm
```

When finished type exit to release the allocation.

How to allocate 1 compute node with 4 cores in the interactive partition with 4 xterm shells:

```
> salloc -N1 -n4 -p interactive

> srun --x11=first --pty xterm
```

Note: the four xterms may be placed in a stacked fashion. This behavior is xserver defined.

Or obtain a bash shell:

```
> salloc -N1 -n1 -p interactive
salloc: Granted job allocation 731

> srun   --pty     --x11=first   bash
sh-4.1$ hostname
alpha021

sh-4.1$ ......do more things......
sh-4.1$ exit
```

After the exit, you are back on the login node.

```
> hostname
login.cluster.tufts.edu
>
```

Comments on potential errors with threaded parallelism

Sometimes you might see errors involving an application where the errors are related to threads, sockets, and other odd messages. Often threaded applications are just fine if there is one thread per cpu. The slurm cpu definition counts enabled hyperthreads as cores/cpus. The slurm batch partition has compute nodes with and without hyperthreads enabled. The following list of nodes are **not** hyperthread enabled. To include this list in your sbatch options, your job will land on a node in the list. This approach often cures this problem. There are other approaches if you need to use Cisco nodes.

```
--constraint=M3
```

Workflow/job pipeline dependent jobs

This is a simple example of dependent jobs. Suppose you want to set off 12 slurm jobs each with 10 tasks(matlab, fortran, C++) using a slurm **job array** in your script so that they are placed nearly all at once onto compute nodes. And you want each of the remaining 11 jobs to depend sequentially on the one before it.

```
For example, set off the first job of 10 tasks:
> sbatch your_job_script_1.sh

slurm will then report the job id: 123456789

then submit the second batch of 10:
> sbatch --dependency=afterok:123456789 your_job_script_2.
sh

> .... etc.....
```

See the manual page for sbatch **--dependency=** options which takes several arguments. Several types of job conditions/options may be specified to fit your needs.

Interactive and gui applications

The srun command without a preceding salloc command is a one step default allocation. After commands are issued, typing exit will release the allocation. The default allocation may or may not meet your needs.

To run a bash shell in the batch partition without X window forwarding support:

> *srun --pty -p batch bash -i*

To run a bash shell in the interactive partition:

> *srun --pty -p interactive bash -i*

How to specify a qos request:

> *srun --pty -p batch --account=normal --qos=normal bash*

Abaqus:

```
> module load abaqus/6.11-2

> srun --pty --x11=first -c 4  -p interactive abq6112.exe   cae -mesa
```

Note that there are several versions of Abaqus installed and the executable may or may not have the **.exe** extension.

```
> module load abaqus/6.14-1

> srun --pty --x11=first -c 4  -p interactive abq6141   cae -mesa
```

The OpenGL -mesa switch/option may or may not be needed depending on your xserver installed on your machine.

Note: for larger abaqus model mesh nodes and the potential to use more memory, you should adjust access by unlimiting your shell's limits:

```
> ulimit –s unlimited
```

Do this prior to running abaqus.

Ansys/Fluent:

```
module load ansys
> srun --pty --x11=first -p batch -c 4   --time=0-2:00:00   fluent
2d
...or....

> srun --pty --x11=first -p interactive  -c 4 fluent 3d

...or....

> srun --pty --x11=first -p interactive  -c 4 icepak
```

Note that you must load Ansys since Fluent products are part of Ansys.

Matlab:

How to run a compiled Matlab program, that you compiled using Matlab's compiler, interactively using 4 threads:

```
-bash-4.1$ salloc -N1 -c4 -p interactive

salloc: Granted job allocation 33

bash-4.1$ module load MCR/R2012a

bash-4.1$ srun ./inverse_limits

ans =

  1.2000e+04

Elapsed time is 54.520722 seconds.

bash-4.1$ exit

exit

salloc: Relinquishing job allocation 33

-bash-4.1$
```

To run a matlab gui session in the interactive partition and request a ten thread allocation:

```
bash-4.1$  salloc -N1 -c4  -p interactive

salloc: Granted job allocation 127

bash-4.1$ module load matlab

bash-4.1$ srun --x11=first --pty  matlab

.... presto!  ..... then exit matlab the normal way and don't forget to relinquish the allocation with either exit or ctrl-d

bash-4.1$  exit
```

Comments on Matlab Parallel options:

- **Matlab parallism is supported with threads or mpi processes**
- **By default matlab will use threads, no special code changes are needed by you**
- slurm will strictly enforce your requested allocation
- not all Matlab functions are threaded or mpi enabled
- Matlab thread parallelism is limited to a node
- Matlab mpi parallelism can be limited to a node or distributed across nodes
- Matlab mpi parallelism requires changes to your code
- Matlab mpi access requires a Matlab Parallel configuration that you set within matlab
- and.... your slurm request must match
- use matlab/2015b or newer for mpi parallelism
- a maximum of 32 cores/cpus per mpi job is available under current license
- M3 nodes in the batch partition have 12 cores each
- M4 nodes have 16 cores and Cisco nodes have 20 cores each for mpi

Single node Matlab MPI jobs

The previous examples use matlab threads. To use mpi parallelism, you must create a parallel profile in Matlab. Those settings are stored in your home directory in a subdirectory called

.matlab/R2015b/parallel.settings

You can create several named profiles which will be stored in **parallel.settings**. Consider creating profiles for 12,16, 20 and 32 cores. Name them separately. Within Matlab, look for the **Parallel** option on the **HOME tab** in the **Environment** section. Choose **Manage Cluster Profiles** and fill accordingly and save. For example, you create a profile with name, MATLAB-12, to define 12 cores. Make one the default. Close Matlab.

If you have a matlab code using matlab **mpi parallel language directives**, such as **parfor,** then you may run matlab on 1 node with 12 cores in the following manner:

```
-bash-4.1$ module load matlab/2015b

-bash-4.1$ srun --pty --x11=first -n 12  -p interactive matlab
```

Note, it is also possible to use the batch partition instead of interactive, but placement may take longer.

Distributed MPI Matlab jobs

Matlab's MDCS software is needed to allow for distributed mpi jobs. This is already setup on compute nodes, but requires you to copy a small directory into your home directory.

```
> cp -a  /cluster/tufts/rt/mpi_matlabr2015b/   .
Then within matlab, add this location to your Matlab path.
```

```
Then to run with a pool of 16 cores across 4 nodes for 2 hours, start Matlab:

> srun --pty --x11=first -N 4  -n 16  -p mpi   --time=0-2:00:00   matlab
```

Follow the examples in the Parallel Computing Documentation for related matlab **pool** operations.

Note: It may be useful to open the **Manage Cluster Profiles** tool and **Validate** a parallel profile as a test.

Maple:

To run the graphical interface to Maple using 4 threads:

```
-bash-4.1$ salloc -N1 -c4  -p interactive

salloc: Granted job allocation 279


bash-4.1$ module load maple

bash-4.1$ srun   --pty  --x11=first  xmaple

bash-4.1$ exit
```

Ansys:

Ansys has several gui interfaces, the launcher145 is just one.

```
-bash-4.1$ salloc -N1 -c4  -p interactive

bash-4.1$ module load ansys/14.5

bash-4.1$ srun --pty   --x11=first  launcher145

bash-4.1$ exit
```

SAS:

```
-bash-4.1$ salloc -N1 -c 4 -p batch  --time=0-2:00:00  bash
salloc: Granted job allocation 72250

bash-4.1$ hostname
login001
bash-4.1$ module load SAS/9.3
bash-4.1$ module list
Currently Loaded Modulefiles:
  1) SAS/9.3


bash-4.1$ srun --pty --x11=first -p interactive sas_u8

Don't forget to exit your allocation.  If you don't the allocation will be killed after 2 hours.

bash-4.1$ exit
```

NOTE: Interactive use of SAS may require that you close your desktop browser(s) in order to display SAS HTML and ODS graphics. These options are altered in the SAS Preferences pull-down menu system.

Mothur:

Interactive use of mothur should be scaled to reflect your in-memory data requirements. As an example, the following will place your job on one of the interactive nodes:

```
bash-4.1$  salloc  -c4 --mem 16000 -p interactive
bash-4.1$  srun  --pty --x11=first  bash
bash-4.1$ module load mothur
bash-4.1$ mothur

.....
>mothur> quit()
bash-4.1$  exit
```

The exit is needed to release the allocation and to get back to the login node of the cluster.

Stata:

Stata is best run from a bash shell.

```
bash-4.1$  module load stata/13

bash-4.1$  srun --account=normal --pty --x11=first  -p interactive --time=0-2:00:00   bash

bash-4.1$ xstata
```

Rstudio:

This is an interactive session using the these versions of R and Rstudio.

```
>  module load RStudio/0.98
> module load R/3.1.0
>  srun --pty --x11=first -p interactive  rstudio
```

Note: R and RStudio does not by default make use of threads. Support for thread parallelism is in the R   parallel package. Your explicit loading of it is required along with the necessary coding changes supporting your tasks. If you include -c slurm option requesting cores, your use of the parallel package options must match the cores requested.

VMD:

Suppose you would like to work out of a bash shell and you have x11 graphics support:

```
> srun -p interactive  --pty --x11=first bash
> module load vmd
> vmd
```

Batch examples:

COMSOL in batch mode using the sbatch option --wrap:

This large memory example requests from both slurm and comsol 8 threads. The number that works best is application and solver dependent.

```
> module load comsol/4.4
>  sbatch -p largemem -c 4 --mem 40000 --time=0-2:00:00  -o my_out.txt  --wrap='comsol -np 4 batch --inputfile round_jet_burner.mph --
outputfile jet_out.mph batchlog jet.log'
```

Make sure both thread arguments, 4, matches and set the expected duration time of your job.

ABAQUS in batch mode:

Create a text file script with content such as the following. Note that the counter-intuitive option "interactive" is an abaqus option. Check the docs to decide if you need this option. The input is in the .imp file format containing data, model, materials,etc...

```
-bash-4.1$ cat my_abaqus_batch.sh
```

```
#!/bin/sh
#SBATCH -J abaqus_batch
#SBATCH -p batch
#SBATCH  --time=0-2:00:00
#SBATCH c 4  --mem=12
#SBATCH --ntasks-per-core=1
#SBATCH --output=abaqus_batch.%N.%j.out
#SBATCH --error=abaqus_batch.%N.%j.err
module load abaqus/6.11-2
module list
date

ulimit –s unlimited

unset SLURM_GTIDS


# comment....  abaqus batch Run with threaded parallelism
abq6112.exe mp_mode=threads cpus=4 memory="6 gb" job=testjob input=abaqus-cluster scratch=/scratch interactive
```

Note: some documentation suggests that your slurm request for memory should be double what is asked for in an Abaqus run. Under some conditions this can matter.

Now, to run it:
```
> sbatch my_abaqus_batch.sh
```

But what if I need to use a fortran subroutine called, my_test.f with this job. How do I do that?

```
#!/bin/sh
#SBATCH -J abaqus_batch
#SBATCH -p batch

#SBATCH --time=0-2:00:00
#SBATCH -N 1
#SBATCH -c 4
#SBATCH --ntasks-per-core=1
#SBATCH --output=abaqus_batch.%N.%j.out
#SBATCH --error=abaqus_batch.%N.%j.err

module load ifc

module load abaqus/6.11-2
module list
date

ulimit –s unlimited

  unset SLURM_GTIDS


# comment....  abaqus batch Run with threaded parallelism
abq6112.exe mp_mode=threads cpus=4 job=testjob input=abaqus-cluster scratch=/scratch  user=my_test.f  interactive
```

ANSYS in batch mode:

Assuming you have a model that is saved as a .dat ansys file, you can use it as input. The file below was taken from the vendor set of examples.

```
>  find  /opt/shared/ansys_inc/v150/v150/ansys/    -name vm233.dat   -type f -print

/opt/shared/ansys_inc/v150/v150/ansys/data/verif/vm233.dat

>  cp   /opt/shared/ansys_inc/v150/v150/ansys/data/verif/vm233.dat   vm233test.dat
```

Create a sbatch script file called ansys_batch.sh with a text editor(nano, nedit, vi, vim...) and use the following content.

Note: this example constrains the job to run on M3 nodes. These nodes have hyperthreads turned off; unlike the Cisco nodes.

```
#!/bin/sh

#SBATCH -J my_ansys

#SBATCH -p batch

#SBATCH -N 1


#SBATCH --constraint=M3

#SBATCH --ntasks-per-core=1

#SBATCH --output=ansys_batch.%N.%j.out

#SBATCH --error=ansys_batch.%N.%j.err

# load the specific version needed

module load ansys/15

module list

date

# comment....   batch Run with threaded parallelism

sbatch --time=0-2:00:00 -c 4 --mem=8g  --wrap="ansys150  -np 4 -j vm233 -b  < vm233test.dat"
```

This job is then submitted to slurm:

> chmod 755 ansys_batch.sh

> ./ansys_batch.sh

If successful, your directory should contain the following files:

```
ansys_batch.sh      magsolv.out  SCRATCH          vm233.db    vm233.err   vm233.full  vm233.log   vm233.r001  vm233.rmg      vm233.vrt
ansys_batch.sh.bck  scratch      slurm-9062219.out  vm233.emat  vm233.esav  vm233.ldhi  vm233.osav  vm233.rdb   vm233test.dat
```

Maple as batch example

Suppose you have a simple maple code file as a text file named maple_test3:

```
p := x^2-x-2;

q := (x+1)^2;

s := (x^2-x-2)/(x+1)^2;

result := s / p * q;
```

To run this file as a slurm batch job:

```
> module load maple

>sbatch -N1 -c2  -t 30  --mem 4G -p batch  --wrap="maple < maple_test3 > maple_test3.output"
```

Note: your actual needs for cores and memory and time need to be specified.

The output stored in maple_tes3.output is:

```
-bash-4.1$ cat maple_test3.output
    |\^/|     Maple 2016 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2016
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> p := x^2-x-2;
                                  2
                             p := x  - x - 2

> q := (x+1)^2;
                                        2
                              q := (x + 1)

> s := (x^2-x-2)/(x+1)^2;
                                  2
                                 x  - x - 2
                             s := ----------
                                        2
                                  (x + 1)

> result := s / p * q;
                               result := 1


>
> quit
memory used=0.4MB, alloc=8.3MB, time=0.06
```

SAS in "interactive-batch" mode example without a GUI:

```
> module load SAS/9.3
> sbatch  -c 4 -p batch --time=0-2:00:00  --mem 8g  sas_u8    your_file_of_commands.sas
```

If successful you should have a .log and .lst file with contents.

Stata in batch mode using 4 threads(large memory example requesting 16 Gig):

```
> salloc --mem=16000 -c 4 -p batch
> module load stata
> sbatch   --time=0-2:00:00  --wrap="stata-mp -b your_stata_command_file.do"
```

Simple R example for running many R programs as a slurm array

Suppose you have a directory named serial_R_example, with five R scripts(could be 500) and you wish to run them somewhat in "parallel" but as separate jobs so that you don't have to wait for them to finish serially. Each file may have different input conditions, or even different goals.   Use the **slurm array** feature. Create an sbatch script, say **my_big_job.sh**, that has the following tasks:

```
#!/bin/bash

#SBATCH -p batch -c 1  --mem=1000

#SBATCH --time=0-2:00:00

#SBATCH  --array=1-5   -n 1  --ntasks-per-node=1

#SBATCH  --error=testjob.err --output=mytestjob.output

#SBATCH  --account=normal --qos=normal

module load R/3.1.0

FILES=$HOME/serial_R_example/*.r

for f in $FILES ; do

R  --no-save <  $f   > $f.output

sleep 2

done
```

The above represents a request for minimal   resource needs, but that can be changed to reflect your needs.   Then to submit this to the cluster:

> **sbatch my_big_job.sh**

Each of the five jobs will be launched on different nodes and your R output will be written to the same directory as the R scripts. There are many slurm approaches to do the same thing.

Note: there is a tool called **mktemp** that assists in the temporary creation of files following a template/naming convention. This could be useful for simulation runs involving many files. For details see:

> man mktemp

Note: Other programs such as comsol, abaqus, mathematica, VMD, etc..., are addressed similarly for interactive or batch use. Executable names may be different from marketing names, as is the case on the old cluster.

More Script examples

Example 1: Suppose a text file named test_batch.sh contains the following three lines:

```
#!/bin/sh

echo begin_test

srun -o out_%j_%t hostname
```

This will produce output files with the name of the compute node that the hostname command was executed on. srun output file out has the jobnumber and (task/step/rank) number in the file name.

To submit it to run on 160 cores:

```
> sbatch -p batch --time=0-0:20:00  -n160 -o out_%j  ./test_batch.sh

> ls  out_327*
```

```
out_327      out_327_121  out_327_146  out_327_27  out_327_51  out_327_76

out_327_0    out_327_122  out_327_147  out_327_28  out_327_52  out_327_77

out_327_1    out_327_123  out_327_148  out_327_29  out_327_53  out_327_78

.................. snip .......
```

Example 1a:

Suppose you wish to submit a single Matlab batch job all on one command line where you need access to 16gig of ram and 8 threads:

The key is the --wrap option that bundles all arguments for the sbatch invocation.

```
> sbatch --mem=16000 -c 8 -p batch   --time=0-2:00:00   --wrap="matlab -nodisplay < your_matlab.m"
```

Example2:

Suppose you have a text file called slurm-numbers.sh   with the following lines:

```
#!/bin/bash

#SBATCH --partition=batch

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=1

#SBATCH --mem=100

#SBATCH --time=0-0:02:00

#SBATCH --output=slurm-numbers.%N.%j.out

#SBATCH --error=slurm-numbers.%N.%j.err

#SBATCH --mail-user=$USER

for i in {1..100};

do echo $RANDOM >> RandomNumbers.txt

done

sort RandomNumbers.txt
```

Note: --time= is an option for setting the max job duration of 2 minutes. If this is the case, then slurm will use that info to help expedite job placement. It is important to guess the max time accurately because if the job exceeds that limit it will be killed.

To run this as a batch job use sbatch command.

Make the file executable and submit:

```
> chmod 755 slurm-numbers.sh

> sbatch ./slurm-numbers.sh
```

When the batch job finishes three resulting files will be written into your current directory.

File RandomNumbers.txt will have 100 sorted intergers. The other two files will have nodename and jobID appended to the root name provided.

Example 3:

Simple Matlab script and batch example

1) Create a text file, example.m, with matlab commands:

```
echo

diary example.output    % this will store .m file output in file example.output.

load xy.dat             % load a flat ascii life containing 2 columns into xy.

xy                      % print  the matrix xy.

x=xy(:,1)           % extract column 1 and call it x.

y=xy(:,2)

dot=x'*y

dot

a = [ 1 2.4 3.9; 4.5 5 6.3; 7.3 8 19 ]

for i = 1:3,

  for j = 1:3,

   b(i,j) = 3*(i-1)+j;

  end

end

[l,u]=lu(a)

A=l*u

save lu.out  l  u -ascii   % save only the l and u matrix in lu.out.

c = (a*b)-b

b

d=inv(c)

who     % what are the matrices available.

exit    % exit matlab is optional.
```

2) Create a shell script, matlab_script.sh , with the following and make it executable with chmod:

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --nodes=1 --time=0-1:00:00

#SBATCH -c 4
#SBATCH --output=matlab.%N.%j.out
#SBATCH --error=matlab.%N.%j.err
module load matlab
matlab -nojvm -nodisplay < example.m
```

3) submit as a batch job requesting 8 threads:

```
> sbatch  ./matlab_batch.sh
```

Example 4: Simple job array script and batch example

Suppose you wish to run several instances(say 4) of a program so that they run on different nodes. The following sbatch script will provide this allocation.

Create a file: myprogram_array.sh with the following content. Change mem and time as needed:

```
#!/bin/bash
#SBATCH --time=5:00
#SBATCH --partition=batch
#SBATCH  --nodes=1
#SBATCH -n 1
#SBATCH --array=1-4
#SBATCH --mem=6000
#SBATCH --ntasks-per-node=1
#SBATCH --output=my_array.%N.%j.out
#SBATCH --error=my_array.%N.%j.err
module load  your_needed_module(s)
./your_executable
```

Note: --time=5 is an option for setting the max job duration of 5 minutes. If this is the case, then slurm will use that info to help expedite job placement by not assuming the job will take the partition time duration limit. It is important to guess the max time accurately because if the job exceeds that limit it will be killed.

```
-bash-4.1$ sbatch  myprogram_array.sh
Submitted batch job 3324389

-bash-4.1$ squeue -u your_utln
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
         3324389_1     batch   myprogram_array   utln  R       0:09      1 m3n01
         3324389_2     batch  myprogram_array   utln  R       0:09      1 m3n02
         3324389_3     batch myprogram_array     utln   R       0:09      1 m3n03
         3324389_4     batch  myprogram_array    utln   R       0:09      1 m3n05
```

More scripting examples can be found on the slurm website and elsewhere.

Example 5:

Batch script passing variables to Matlab program with an expected runtime of 4 hours.

Create a text file, matlab_parametric_input.sh with the following contents. The goal is to submit 12 matlab jobs with 12 different pairs of inputs to a matlab program.

```
#!/bin/bash

#SBATCH --mem=4g

#SBATCH  --time=0-4:00:00

#SBATCH --partition=batch

#SBATCH -n 1

#SBATCH --array=1-12

#SBATCH --output=slurmoutput.%N.%j.out

#SBATCH --error=slurmerrout.%N.%j.err

module load matlab

date

RADIUS123="5 15"

YEAR123="1960 1970 1980 1990 2000 2010"

for radius in `echo $RADIUS123`;

do

for year in `echo $YEAR123`;

do

echo "$radius"

echo "$year"

sbatch --wrap='matlab -nodisplay -nodesktop -nosplash -r "m_param('$year','$radius'); exit"'

done

done
```

The simple matlab program being submitted is named m_param.m with content:

```
% catch the input environment vars

function test_param(year, radius)

year

radius

sum=year + radius;

sum

who

exit
```

Example not using slurm array features:

Suppose you wish to run many files, perhaps R scripts, that are located in some directory, perhaps myscripts/ .   Create a text file, runBatchR.sh with the following commands. Package up your sbatch options in the variable, opts. Some options like --mail are optional. Make an accurate time assessment! And make sure your R scripts end in a number, for example your_file_5.R, etc... Create a file list text file, myfilelist.txt, with the names of the R files.

```
#!/bin/bash
cd  $HOME/myscripts/
module load R/3.2.5
opts="-p batch -c 4 --mem=10000 --time=10:00:00"
while read filenm; do
   echo $filenm
      outs="--output=$filenm.out --error=$filenm.%N.err --mail-type=ALL --mail-user=$USER"
      sbatch $opts $outs --wrap="R --no-save < $filenm > $filenm.output"
      sleep 2
Done
```

To change the permission and run the script:

```
> chmod 755 runBatchR.sh
>  cat myfilelist.txt  | sh runBatchR.sh
```