Transitioning to SLURM from Maui/Torque

THE Grid Feup is changing the cluster to utilize Slurm as its scheduler.

There will be some differences to the current setup (Torque/Maui), but it is not a big learning curve.

We also provide a wrapper for the Torque commands to run on this Slurm cluster so you can try submitting your current script to the new cluster and notice the difference.

Slurm website can be found here: http://slurm.schedmd.com/

- Guide to SLURM from LLNL
- Summary: http://slurm.schedmd.com/pdfs/summary.pdf
- Tutorials: http://slurm.schedmd.com/tutorials.html
- Documentation: http://slurm.schedmd.com/documentation.html
- Comparison between PBS/TORQUE & SLURM: http://slurm.schedmd.com/rosetta.html
- Exit Codes

The outline of the command differences between Slurm and Torque are available at http://slurm.schedmd.com/rosetta.html & http://slurm.schedmd.com/documentation.html

Important Notes

- The current scheduler (Maui/Torque) will be decommissioned by the end of September 2019. So, start transitioning to SLURM - ssh to slurmsub.grid.fe.up.pt (via submit.grid.fe.up.pt) .
- The slurm script generator can be found at https://grid.fe.up.pt/web/index.php?page=slurm_job_script_generator .
- The wrappers have been created. So, most of the torque commands (such as qsub) that you are used to should work fine. However, please start mastering SLURM commands.
- The same PBS batch script can be used with the addition of the line “#!/bin/bash” on the top of the file in most of the cases. See the SLURM Environment variable section below.
- For complicated job submission, we recommend you to understand the SLURM syntax.
- SLURM command “srun” does not seem to work properly when used within MPI context with the following error. So, don't use srun for parallel jobs.

```
srun -n 4 hello_mpi
srun: error: Unable to create job step: More processors requested than
permitted
```

Some more details (adapted from ACCRE documentation) [1]:

1. Introduction

SLURM (Simple Linux Utility for Resource Management) is a software package for submitting, scheduling, and monitoring jobs on large compute clusters. This page details how to use SLURM for submitting and monitoring jobs on Grid Feup cluster. New cluster users should consult our Getting Started pages, which is designed to walk you through the process of creating a job script, submitting a job to the cluster, monitoring jobs, checking job usage statistics, and understanding our cluster

policies.

Grid currently used Torque for resource management and Maui for job scheduling, and users submitted a job to Grid Feup by writing a script specifying the resources and commands needed to execute a program. SLURM also requires users to submit jobs through a script, with slightly different syntax compared to Torque/Maui. These differences are highlighted in section 2. A summary of SLURM commands is shown in section 3. (A great reference for SLURM commands can also be found by clicking here.)

2. SLURM vs. Torque

2.1. Job Submission

2.1.1. Batch

Converting a Torque batch script to SLURM is generally a straightforward process. Below is a simple SLURM script (right hand side) for running a Matlab job requesting 1 node, 1 CPU core, 500 MB of RAM, and 2 hours of wall time. For comparison, the equivalent Torque script is shown on the left. Aside from syntax, the two scripts have only very minor differences. In general, **#SBATCH** options tend to be more self-explanatory. Note that specifying the node (**#SBATCH –nodes=1**) and CPU core (**#SBATCH –ntasks=1**) count must be broken off into two lines in SLURM, and that SLURM has no equivalent to **#PBS -j oe** (SLURM combines standard output and error into a single file by default).

Torque/Maui script - simple.pbs	**Slurm script - simple.slurm**
	#!/bin/bash
#PBS -M your@email.address	#SBATCH –mail-user=your@email.address
#PBS -m bae	#SBATCH –mail-type=ALL
#PBS -l nodes=1:ppn=1	#SBATCH –nodes=1
	#SBATCH –cpus-per-task=1
#PBS -l walltime=2:00:00	#SBATCH –time=2:00:00
#PBS -l mem=500mb # Note: mem is total memory per job	#SBATCH –mem=500m # Note: –mem is memory per node
#PBS -o matlab_job.out	#SBATCH -o matlab_job.o%j
#PBS -j oe	
cd $PBS_O_WORKDIR	
cp -r * $PFSDIR	cp -r * $PFSDIR
cd $PFSDIR	cd $PFSDIR
module load matlab	module load matlab
matlab -singleCompThread -nodisplay -r fixedMC	matlab -singleCompThread -nodisplay -r fixedMC
cp * $PBS_O_WORKDIR/.	cp * $SLURM_SUBMIT_DIR

A SLURM batch script must begin with the **#!/bin/bash** directive on the first line. The subsequent lines begin with the SLURM directive **#SBATCH** followed by a resource request or other pertinent job information. Email alerts will be sent to the specified address when the job begins, aborts, and ends.

Submit the batch job:

```
sbatch simple.slurm #slurm
qsub simple.pbs     #PBS
```

(Note that the file extensions .pbs and .slurm are only for convenience.)

2.1.2. Interactive

srun can be used to run interactive jobs, with or without graphics

```
srun --x11 -n 1 -c 2 --time=1:00:00 --pty /bin/bash
```

Note: This will launch two cpus per task (-c 2) with a single task (-n 1) for 1 hour (–time=1:00:00), with graphical windows (–x11) ready. If you want to request, let's say, 100 hrs, you can do it as –time=4-04:00:00 in the the format: “days-hours:minutes:seconds”.

For GPU, it becomes:

```
srun --constraint=GPU -n 1 -c 2 --time=1:00:00 --mem=5gb --pty /bin/bash
```

Note: This will request a node with gpu feature (–constraint=GPU) with memory-per-node 5gb (–mem=5gb).

For reference, the following table lists common Torque options along side the equivalent option in SLURM. For examples of how to include the appropriate SLURM options for parallel jobs, please refer to Section 4.

Request a Specific Node:

```
srun --nodelist=avafat01 --pty bash
```

It is requesting avafat01. It is useful when we want to use a specific node only. Please be mindful that you might have to wait until the node is free to use if someone else is running a job in that node.

Exclude a particular node or multiple nodes from running your jobs:

```
srun --exclude=ava01,ava02 --pty bash
```

If you want to exclude many nodes, create a file (e.g. ava01-10) with the nodelist as showed:

```
ava01,ava02,ava03,ava04,ava05,ava06,ava07,ava08,ava09,ava10
```

And run it as:

```
srun --exclude=./ava01-10 --pty bash
```

If you want to reserve the whole 12 processors comp node, you can use, hex keyword (hex⇒ 6 cores; 12 processors): srun -n 12 -C hex24gb –pty bash

For 16 processors comp node, use octa (octa ⇒ 8 cores; 16 processors):

```
srun -n 12 -C octa64gb --pty bash
```

You need to specify the queue type (e.g. batch, int) to use the resources available in nodes in those queue. For example, to request 32 processors in a node, you need to use the nodes in batch queue (-q batch).

```
srun -p batch -n 32 --mem=5gb --pty bash
```

Command	Torque/MAUI	Slurm
submit command	qsub	sbatch, srun, salloc
walltime request	#PBS -l walltime hh:mm:ss	#SBATCH –time=hh:mm:ss (or -t hh:mm:ss)
specific node request	#PBS -l nodes=X:ppn=Y:gpus=2	#SBATCH –nodes=X (or -N X) #SBATCH –cpus-per-task=Y(or -c Y)[for OpenMP or hybrid code] #SBATCH –ntasks-per-node=Y(or -n Y)[to set equal # of tasks/node or MPI] #SBATCH –n Z [set total number of tasks] #SBATCH –gres=gpu:2
define memory	#PBS -l mem=Zgb	#SBATCH –mem=Zgb
define number of procs/node		#SBATCH -c <# of cpus/task> → for OpenMP/hybrid jobs #SBATCH -n <# of total tasks or processors → for MPI jobs
queue request	#PBS -q batch	#SBATCH -p batch
group account	#PBS -A <account or GorupID>	#SBATCH -A <account or GroupID>
job name	#PBS -N <name>	#SBATCH -J <name>
output file name	#PBS -O <filename>	#SBATCH -o <name>.o%j where %j is the jobID
email option	#PBS -m e	#SBATCH –mail-type=end options: begin,end,fail,all
email address	#PBS -M <email address>	#SBATCH –mail-user=<email>
count processors	NPROCS=`wc -l < $PBS_NODEFILE	NPROCS=$(($SLURM_NNODES * $SLURM_CPUS_PER_TASK))**#for OpenMP & hybrid (MPI + OpenMP) jobs** NPROCS=$SLURM_NPROCS or $SLURM_NTASKS **#for MPI jobs** **Note**: -N 4 -n 16 ⇒ 16 processors NOT 4*16=64 processors **if $PBS_NODEFILE is needed, then include the following lines:** PBS_NODEFILE=`generate_pbs_nodefile` NPROCS=`wc -l < $PBS_NODEFILE `
starting directory on the compute node	user home directory	the working (submit) directory

Command	Torque/MAUI	Slurm
interactive job request	qsub -I -X	srun --x11 --pty /bin/bash
Reserve the node exclusively		srun --exclusive --pty /bin/bash
dependency	#PBS -d <jobid>	#SBATCH -d <after:jobid>

2.2. Important Examples

2.2.1. Account

Maui/Torque:

```
#PBS -A <caseID>_share
```

SLURM:

```
#SBATCH -A <caseID>
```

2.2.2. Processor count

MPI Job:

Request 2 nodes (-N 2) with total 4 processors (-n 4)

```
srun -N 2 -n 4 --pty bash
```

Use Slurm env variable "$SLURM_NTASKS" to get the total number of processors

```
NPROCS=$SLURM_NTASKS
```

Check

```
echo $NPROCS
4
```

(Note: -n gives the number of processors. So, 4 processors (-n 4) from two nodes (-N 2)

OpenMP or Hybrid Job:

Request two tasks (-N 2) with 4 processors per task (-c 4)

```
srun -N 2 -c 4 --pty bash
```

Use Slurm env variables "$SLURM_NTASKS & $SLURM_CPUS_PER_TASK" to get the total number of processors.

```
NPROCS=$(( $SLURM_NTASKS*$SLURM_CPUS_PER_TASK ))
```

Check:

```
echo $NPROCS
8
```

Note: Two tasks (-N 2) with 4 CPUS/processors per task (-c 4) gives total of 2*4=8 processors.

3. SLURM Commands

Just like Torque, SLURM offers a number of helpful commands for tasks ranging from job submission and monitoring to modifying resource requests for jobs that have already been submitted to the queue. Below is a list of SLURM commands, as well as the Torque equivalent in the far left column.

Torque	SLURM	Function
qsub [job_script]	sbatch [job_script]	Job submission
qstat or showq	squeue	Job/Queue status
qdel [JOB_ID]	scancel [JOB_ID]	Job deletion
pbsnodes	scontrol show nodes	Node list
qhold [JOB_ID]	scontrol hold [JOB_ID]	Job hold
qrls [JOB_ID]	scontrol release [JOB_ID]	Job release
qstat -a	sinfo	Cluster status
qsub -I	salloc	Launch interactive job
	srun [command]	Launch (parallel) job step
	sacct	Displays job accounting information

Similar command to “qstat”

```
squeue -u <caseID>
```

3.1. sbatch

The sbatch command is used for submitting jobs to the cluster. Like Torque’s qsub, sbatch accepts a number of options either from the command line, or (more typically) from a batch script. An example of a SLURM batch script (called simple.slurm) is shown below:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 1
#SBATCH --mem-per-cpu=1G
#SBATCH --time=0-00:15:00     # 30 minutes
#SBATCH --output=my.stdout
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=ALL
#SBATCH --job-name="just_a_test"

# Put commands for executing job below this line
# This example is loading Python 2.7.8 and then
```

```
# writing out the version of Python
module load python
python --version
```

To submit this batch script, a user would type:

```
sbatch simple.slurm
```

This job (called just_a_test) requests 1 compute node, 1 task (by default, SLURM will assign 1 CPU core per task), 1 GB of RAM per CPU core, and 15 minutes of wall time (the time required for the job to complete). Note that these are the defaults for any job, but it is good practice to include these lines in a SLURM script in case you need to request additional resources.

Optionally, any #SBATCH line may be replaced with an equivalent command-line option. For instance, the #SBATCH –ntasks=1 line could be removed and a user could specify this option from the command line using:

```
sbatch --ntasks=1 simple.slurm
```

The commands needed to execute a program must be included beneath all **#SBATCH** commands. Lines beginning with the # symbol (without /bin/bash or SBATCH) are comment lines that are not executed by the shell. The example above simply prints the version of Python loaded in a user's path. It is good practice to include any ***setpkgs*** commands in your SLURM script. A real job would likely do something more complex than the example above, such as read in a Python file for processing by the Python interpreter.

For more information about sbatch see: http://slurm.schedmd.com/sbatch.html

3.2. squeue

squeue is used for viewing the status of jobs. By default, squeue will output the following information about currently running jobs and jobs waiting in the queue: Job ID, Partition, Job Name, User Name, Job Status, Run Time, Node Count, and Node List. There are a large number of command-line options available for customizing the information provided by squeue. Below are a list of examples:

Command	**Meaning**
squeue –-long	Provide more job information
squeue –-user=USER_ID	Provide information for USER_ID's jobs
squeue –-account=ACCOUNT_ID	Provide information for jobs running under ACCOUNT_ID
squeue –-states=running	Show running jobs only
squeue –-Format=account,username,numcpus,state,timeleft	Customize output of squeue
squeue –-start	List estimated start time for queued jobs
squeue –-help	Show all options

For more information about squeue see: http://slurm.schedmd.com/squeue.html

Similar command to "showq" or "qstat"

```
squeue -u jbarber
```

output:

```
   JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)

   661587     batch     bash   jbarber R      22:21      1 ava10
```

Note the jobID (661587), status of the Job (R→ Running) and the compute node (ava10) that the job is running.

Want to see details such as why your job is in PD (pending) state, in which node your job is running.

```
sq
```

output:

```
730814     batch slurm.sl    bga11 PD       0:00      1   4   6950
(AssocMaxWallDurationPerJobLimi
            730815     batch slurm.sl    bga11 PD       0:00      1   4
6950 (AssocMaxWallDurationPerJobLimi
..
  989833     batch 3DClasse   txh310  R   21:59:56     16 240   3044
comp145t,comp146t,comp147t,comp149t,comp151t,comp154t,comp156t,comp157t,comp
158t,comp159t,comp179t,comp185t,comp186t,comp187t,comp191t,comp192t
            992383     batch job_chec  sxl1036  R    1:43:13      2  16
3007 comp122t,comp123t
```

Also, show the start time and end time of the job which is equivalent to Torque/Maui Showstart

```
squeue -u <CaseID> -o "%.9i %.9P %.8j %.8u %.2t %.10M %.6D %S %e"
```

output:

```
JOBID PARTITION     NAME     USER ST       TIME  NODES START_TIME END_TIME
   676101 batch      JOB   jbarber PD       0:00      1 2018-04-09T15:25:21
   606057 batch      JOB   jbarber  R 8-01:08:45      1 2018-03-31T14:17:02
2018-04-31T14:17:02
   606056 batch      JOB   jbarber  R 8-01:10:16      1 2018-03-31T14:15:31
2018-03-31T14:15:31
```

The job 676101 is estimated to start on April 09 at 15:25 and the end time of job 606057 is April 31 at 14:17.

Filtering squeue output through awk may be useful, for example, to isolate entries with group name in common:

```
squeue -o "%A %C %e %E %g %l %m %N %T %u" | awk 'NR==1 || /eecs600/'
```

output:

```
    JOBID CPUS END_TIME DEPENDENCY GROUP TIME_LIMIT MIN_MEMORY NODELIST
STATE USER

    148137 1 2018-01-26T16:54:22  eecs600 2:00:00 1900 ava01 RUNNING jbarber

    148146 1 2018-01-27T01:14:27  eecs600 10:00:00 1900 ava02 RUNNING
ptsilva
```

Note the jobs status for the users in a group eecs600

3.3. sacct

This command is used for viewing information for completed jobs. This can be useful for monitoring job progress or diagnosing problems that occurred during job execution. By default, sacct will report Job ID, Job Name, Partition, Account, Allocated CPU Cores, Job State, and Exit Code for all of the current user's jobs that completed since midnight of the current day. Many options are available for modifying the information output by sacct:

Command	**Meaning**
sacct –-starttime 12.04.14	Show information since midnight of Dec 4, 2014
sacct –-allusers	Show information for all users
sacct –-accounts=ACCOUNT_ID	Show information for all users under ACCOUNT_ID
sacct –-format="JobID,user,account,elapsed, Timelimit,MaxRSS,ReqMem,MaxVMSize,ncpus,ExitCode"	Show listed job information
sacct –-help	Show all options

The –format option is particularly useful, as it allows a user to customize output of job usage statistics. We would suggest create an alias for running a customized version of sacct. For instance, the elapsed and Timelimit arguments allow for a comparison of allocated vs. actual wall time. MaxRSS and MaxVMSize shows maximum RAM and virtual memory usage information for a job, respectively, while ReqMem reports the amount of RAM requested.

See the status of your job. Note that your executable should be preceded by "srun" command for both serial and MPI executable.

```
sacct -o
JobID,JobName,AveCPU,AvePages,AveRSS,MaxRSSNode,AveVMSize,NTasks,State,ExitC
ode -j <jobID>
```

output:

```
 JobID    JobName     AveCPU   AvePages     AveRSS MaxRSSNode  AveVMSize
NTasks      State ExitCode

------------ ---------- ---------- ---------- ---------- ---------- --------
-- -------- ---------- --------
```

```
1013605        v2o5band
COMPLETED      0:0

1013605.bat+      batch    00:00:00          0      6244K    ava01    308544K
1  COMPLETED      0:0
```

For more information about sacct see: http://slurm.schedmd.com/sacct.html

3.4. scancel

It kills the job. It is the equivalent of Torque/Maui “qdel” command. Example:

```
scancel -i 681457
```

prompt:

```
Cancel job_id=681457 name=bash partition=batch [y/n]? y
srun: Force Terminated job 681457
```

3.5. scontrol

scontrol is used for monitoring and modifying queued jobs. One of its most powerful options is the scontrol show job option, which is analogous to Torque’s checkjob command. scontrol is also used for holding and releasing jobs. Below is a list of useful scontrol commands:

Command	**Meaning**
scontrol show job JOB_ID	Show information for queued or running job
scontrol hold JOB_ID	Place hold on job
scontrol release JOB_ID	Release hold on job
scontrol show nodes	Show hardware details for nodes on cluster
scontrol update JobID=JOB_ID Timelimit=1-12:00:00	Change wall time to 1 day 12 hours
scontrol update dependency=JOB_ID	Add job dependency so that job only starts after JOB_ID completes
scontrol –help	Show all options

Note: The equivalent qstat -f <JOB_ID> & checkjob Maui/Torque command in SLURM is scontrol show job <JOB_ID>

```
scontrol show job 136355
```

output:

```
JobId=136355 JobName=xxxxx

   UserId=xxxx(yyyy) GroupId=xxx(yyy)
```

```
   Priority=3007 Nice=0 Account=gray QOS=normal

   JobState=RUNNING Reason=None Dependency=(null)

   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0

   RunTime=20:07:27 TimeLimit=13-07:00:00 TimeMin=N/A

   SubmitTime=2018-01-18T15:37:55 EligibleTime=2018-01-18T15:37:55

   StartTime=2018-01-18T15:37:56 EndTime=2018-01-31T22:37:56

   PreemptTime=None SuspendTime=None SecsPreSuspend=0

   Partition=batch AllocNode:Sid=hpctest:39249

   ReqNodeList=(null) ExcNodeList=(null)

   NodeList=ava10

   BatchHost=ava10

   NumNodes=1 NumCPUs=8 CPUs/Task=1 ReqB:S:C:T=0:0:*:*

   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*

   MinCPUsNode=1 MinMemoryNode=48G MinTmpDiskNode=0

   Features=(null) Gres=(null) Reservation=(null)

   Shared=OK Contiguous=0 Licenses=(null) Network=(null)

   Command=/home/xxxx/AAA.sh

   WorkDir=/home/xxxx/BBB

   StdErr=/home/xxxx/OOO.o

   StdIn=/dev/null

   StdOut=/home/xxx/OOOO.o

   Power= SICP=0
```

If the job is pending, it will show the reason for pending as well:

```
...
JobState=PENDING
Reason=ReqNodeNotAvail(Unavailable:ava07,ava08,ava09,ava20,ava21,ava22)
Dependency=(null)
```

Here, it shows that the job is waiting for the resources. The gpu featured nodes are listed because they are currently offline.

SLURM command equivalent to pbsnodes <node>

```
scontrol show node ava04
```

output:

```
NodeName=ava04 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=16 CPULoad=15.06
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=ava04 NodeHostName=ava04 Version=17.11
   OS=Linux 3.10.0-693.21.1.el7.x86_64 #1 SMP Wed Mar 7 13:12:24 CST 2018
   RealMemory=68000 AllocMem=0 FreeMem=8298 Sockets=16 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=batch,big
   BootTime=2018-04-20T12:15:29 SlurmdStartTime=2018-11-13T17:23:23
   CfgTRES=cpu=16,mem=68000M,billing=16
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Here, the number of processors (ncpus) is 16, and available Memory (availmem) is 68000 (~ 68gb).

You can also see the slurm configuration info:

```
scontrol show config
```

output:

```
Configuration data as of 2018-03-08T10:39:08
AccountingStorageBackupHost = (null)
AccountingStorageEnforce = associations,limits
AccountingStorageHost   = smaster
...
```

Check the MaxJobCount

```
scontrol show config | grep MaxJobCount
MaxJobCount             = 10000
```

For more information about scontrol see: http://slurm.schedmd.com/scontrol.html

3.6. srun

srun can be used to run interactive jobs, with or without graphics

```
srun --x11 -N 1 -c 2 --time=1:00:00 --pty /bin/bash
```

This will launch two tasks on a single node for 1 hour, with graphical windows ready.

This command can also be used to launch a parallel job step. Typically, srun is invoked from a SLURM job script to launch a MPI job (much in the same way that mpirun or mpiexec are used). More details about running MPI jobs within SLURM are provided below. Please note that your application must include MPI code in order to run in parallel across multiple CPU cores using srun. Invoking srun on a non-MPI command or executable will result in this program being independently run X times on each of the CPU cores in the allocation.

Alternatively, srun can be run directly from the command line on a gateway, in which case srun will first create a resource allocation for running the parallel job. The -n [CPU_CORES] option is passed to specify the number of CPU cores for launching the parallel job step. For example, running the following command from the command line will obtain an allocation consisting of 16 CPU cores and then run the command hostname across these cores:

```
srun -n 16 hostname
```

For more information about srun see: http://www.schedmd.com/slurmdocs/srun.html

3.7 sinfo

sinfo allows users to view information about SLURM nodes and partitions. A partition is a set of nodes (usually a cluster) defined by the cluster administrator. Below are a few example uses of sinfo:

Command	**Meaning**
sinfo –Nel	Displays info in a node-oriented format
sinfo –partition=big	Get information about big nodes
sinfo –states=IDLE	Displays info about idle nodes
sinfo –help	Show all options

Note: If you want to get the detailed options equivalent to “**showq**” and “**mdiag -n**”

```
sinfo -a -o "%P %a %l %D %N %C"
```

output:

```
PARTITION AVAIL TIMELIMIT NODES NODELIST CPUS(A/I/O/T)
int* up 1-00:00:00 12 magalhaes[01-12] 0/68/32/100
```

Here, (A/I/O/T) represents “allocated/idle/other (offline/down)/total”.

Equivalent to “mdiag -n”:

```
sinfo -p batch -Nle -o '%n %C %t'
```

Reasons for possible node failure:

```
sinfo -R
```

For more information about sinfo see: http://slurm.schedmd.com/sinfo.html

3.8. sreport

seport is used for generating reports of job usage and cluster utilization. It queries the SLURM database to obtain this information. By default information will be shown for jobs run since midnight of the current day. Some examples:

Command	**Meaning**
sreport cluster utilization	Show cluster utilization report
sreport user top	Show top 10 cluster users based on total CPU time
sreport cluster AccountUtilizationByUser start=2018-12-01	Show account usage per user dating back to December 1, 2018
sreport job sizesbyaccount PrintJobCount	Show number of jobs run on a per-group basis
sreport –help	Show all options

For more information about sreport see: http://slurm.schedmd.com/sreport.html

3.9. sstat

Display various status information of a running job/step (Refer to SLURM man page).

```
sstat -j <jobID>
```

Very Important: If you are submitting the job using sbatch, please include srun before your executable in your SLURM batch script as showed:

```
srun ./<executable>
```

Selecting the fields of interest

```
sstat -p --format=AveCPU,AvePages,AveRSS,MaxRSSNode,AveVMSize,NTasks,JobID -
j 661587
```

output:

```
AveCPU|AvePages|AveRSS|MaxRSSNode|AveVMSize|NTasks|JobID|

00:00.000|0|2264K|ava20|119472K|1|661587.0|
```

To estimate how much memory is being consumed by run a top comamnd in the node where your job is running.

```
sq | grep <caseID>
```

output:

```
     1958082 batch     Tumor-PIPE-a   <caseID>  R    19:40:29     1   4
1002 ava20
```

```
ssh -t ava20 top
```

output:

```
PID     USER PR NI VIRT RES SHR  S %CPU %MEM TIME+     COMMAND
21348 jxw773 20 0 14.9g 14g 1072 S 400.0 22.6 678:30.15 bwa
```

Note the 22.6% of 64gb comes out to be about 15gb of memory.

4. Parallel Job Example Scripts

Below are example SLURM scripts for jobs employing parallel processing. In general, parallel jobs can be separated into four categories:

- Distributed memory programs that include explicit support for message passing between processes (e.g. MPI). These processes execute across multiple CPU cores and/or nodes.
- Multithreaded programs that include explicit support for shared memory processing via multiple threads of execution (e.g. Posix Threads or OpenMP) running across multiple CPU cores.
- Embarrassingly parallel analysis in which multiple instances of the same program execute on multiple data files simultaneously, with each instance running independently from others on its own allocated resources (i.e. CPUs and memory). SLURM job arrays offer a simple mechanism for achieving this.
- GPU (graphics processing unit) programs including explicit support for offloading to the device via languages like CUDA or OpenCL.

It is important to understand the capabilities and limitations of an application in order to fully leverage the parallel processing options available on the cluster. For instance, many popular scientific computing languages like Python, R, and Matlab now offer packages that allow for GPU or multithreaded processing, especially for matrix and vector operations.

4.1. MPI Jobs

Jobs running MPI (Message Passing Interface) code require special attention within SLURM. SLURM allocates and launches MPI jobs differently depending on the version of MPI used (e.g. OpenMPI, MPICH2, Intel MPI).

```
#!/bin/bash
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=ALL
#SBATCH -N 3
#SBATCH -n 24     # 8 MPI processes per node
```

```
#SBATCH --time=7-00:00:00
#SBATCH --mem=4G     # 4 GB RAM per node
#SBATCH --output=mpi_job_slurm.log

module load openmpi
echo $SLURM_JOB_NODELIST

# Assign the number of processors
NPROCS=$SLURM_NTASKS

#Run the job
mpirun -n $NPROCS ./test
```

This example requests **3 nodes and 8 tasks (i.e. processes) per node, for a total of 24 MPI tasks**. It is recommended for you to just use **“-n” option ONLY** for automatic load balancing. By default, SLURM allocates 1 CPU core per process, so this job will run across 24 CPU cores. Note that mpirun/mpiexec accepts -n <number cpus>). Please avoid using srun command to run parallel jobs at this time, since it does not seem to work well.

Executables generated with older versions of OpenMPI or MPICH2 should be launched using these packages’ native mpirun or mpiexec commands rather than SLURM’s srun. Such programs may run under SLURM but in some cases they may not.

More information about running MPI jobs within SLURM can be found here here: http://slurm.schedmd.com/mpi_guide.html.

4.2. Multithreaded Jobs (OpenMP)

Multithreaded programs are applications that are able to execute in parallel across multiple CPU cores within a single node using a shared memory execution model. In general, a multithreaded application uses a single process (i.e. “task” in SLURM) which then spawns multiple threads of execution. By default, SLURM allocates 1 CPU core per task. In order to make use of multiple CPU cores in a multithreaded program, one must include the –cpus-per-task option. Below is an example of a multithreaded program requesting 4 CPU cores per task. The program itself is responsible for spawning the appropriate number of threads.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --cpus-per-task=4  # 4 threads per task
#SBATCH --time=02:00:00   # two hours
#SBATCH --mem=4G
#SBATCH --output=multithread.out
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=ALL
#SBATCH --job-name=multithreaded_example

#Export Number of Threads
NPROCS=$(( $SLURM_NNODES * $SLURM_CPUS_PER_TASK ))
export OMP_NUM_THREADS=$NPROCS
```

```
# Run multi-threaded application
./hello
```

4.3. Job Arrays

Job arrays are useful for submitting and managing a large number of similar jobs. As an example, job arrays are convenient if a user wishes to run the same analysis on 100 different files. SLURM provides job array environment variables that allow multiple versions of input files to be easily referenced. In the example below, three input files called vectorization_0.py, vectorization_1.py, and vectorization_2.py are used as input for three independent Python jobs:

```
#!/bin/bash
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=ALL
#SBATCH -c 1
#SBATCH --time=2:00:00
#SBATCH --mem=2G
#SBATCH --array=0-2
#SBATCH --output=python_array_job_slurm_%A_%a.out

echo "SLURM_JOBID: " $SLURM_JOBID
echo "SLURM_ARRAY_TASK_ID: " $SLURM_ARRAY_TASK_ID
echo "SLURM_ARRAY_JOB_ID: " $SLURM_ARRAY_JOB_ID

module load python
python < vectorization_${SLURM_ARRAY_TASK_ID}.py
```

The **#SBATCH –array=0-2** line specifies the array size (3) and array indices (0, 1, and 2). These indices are referenced through the SLURM_ARRAY_TASK_ID environment variable in the final line of the SLURM batch script to independently analyze the three input files. Each Python instance will receive its own resource allocation; in this case, each instance is allocated 1 CPU core (and 1 node), 2 hours of wall time, and 5 GB of RAM.

One implication of allocating resources per task is that the node count will not apply across all tasks, so specifying –nodes=1 will not limit all tasks within an array to a single node. To limit the total number of CPU cores (and thus tasks) used simultaneously, use %[CPU_COUNT] following the –array= option. For example, –array=0-100%4 will limit the tasks to running on 4 CPU cores simultaneously. This means the tasks will execute in batches of 4 until all 100 tasks have completed.

The –*array*= option is flexible in terms of the index range and stride length. For instance, –array=0-10:2 would give indices of 0, 2, 4, 6, 8, and 10.

The %A and %a variables provide a method for directing standard output to separate files. %A references the SLURM_ARRAY_JOB_ID while %a references SLURM_ARRAY_TASK_ID. SLURM treats job ID information for job arrays in the following way: each task within the array has the same SLURM_ARRAY_JOB_ID, and its own unique SLURM_JOBID and SLURM_ARRAY_TASK_ID. The JOBID shown from squeue is formatted by SLURM_ARRAY_JOB_ID followed by an underscore and the SLURM_ARRAY_TASK_ID.

While the previous example provides a relatively simple method for running analyses in parallel, it

can at times be inconvenient to rename files so that they may be easy indexed from within a job array. The following example provides a method for analyzing files with arbitrary file names, provided they are all stored in a sub-directory named **data**:

```
#!/bin/bash
#SBATCH --mail-user=your@email.address
#SBATCH --mail-type=ALL
#SBATCH -c 1
#SBATCH --time=2:00:00
#SBATCH --mem=2G
#SBATCH --array=1-5   # In this example we have 5 files to analyze
#SBATCH --output=python_array_job_slurm_%A_%a.out
arrayfile=`ls data/ | awk -v line=$SLURM_ARRAY_TASK_ID '{if (NR == line)
print $0}'`
module load python
python < data/$arrayfile
```

More information can be found here: http://slurm.schedmd.com/job_array.html

8. Torque Wrappers

Torque wrappers are distributed with SLURM to ease the transition from Torque to SLURM. Wrappers are available for virtually all the common Torque commands, including qsub, qstat, qdel, qhold, qrls, and pbsnodes. These wrappers are designed to function in the same way as their Torque counterparts, with support for many of the same options and flags. Therefore, users may be able to run their old Torque scripts without converting them (or with minimal modifications) to SLURM syntax. These jobs will still be managed by SLURM, but to the user it will still “feel” like a Torque environment.

While the Torque wrappers should aid the transition from Torque to SLURM, in the long run we encourage users to convert their job scripts to SLURM. There are a number of reasons for converting to SLURM. The first reason is that native SLURM scripts offer increased flexibility and control over jobs. As the SLURM code base continues to expand, it is unlikely that the Torque wrappers will be fully supported and able to handle more advanced use cases. Troubleshooting and debugging of Torque scripts will also be more difficult.

9. SLURM Environment Variables

Variable	Meaning
SLURM_JOBID	Job ID
SLURM_SUBMIT_DIR	Job submission directory
SLURM_SUBMIT_HOST	Name of host from which job was submitted
SLURM_JOB_NODELIST	Names of nodes allocated to job
SLURM_ARRAY_TASK_ID	Task id within job array
SLURM_JOB_CPUS_PER_NODE	# of procs per node allocated to job (for OpenMP type of jobs)
SLURM_NNODES	Number of nodes allocated to job
SLURM_NPROCS	# of procs allocated to job (for MPItype of jobs)

Each of these environment variables can be referenced from a SLURM batch script using the $ symbol

before the name of the variable (e.g. echo $SLURM_JOBID) A full list of SLURM environment variables can be found here: http://slurm.schedmd.com/sbatch.html#lbAF

References:

[1] ACCRE documentation

[2] HPC University of Maryland

From:
https://grid.fe.up.pt/dokuwiki/ - **GRID FEUP**

Permanent link:
https://grid.fe.up.pt/dokuwiki/doku.php?id=documentation:slurm

Last update: **2020/05/28 11:16**