

Job submission with SLURM

Nanye Long

last update 08/10/2020

Overview

1. HPCC job running policy
2. Writing and submitting SLURM job scripts (: `/mnt/research/common-data/workshops/slurm/`)
3. Checking job status

To learn more:

- SLURM website (`https://www.schedmd.com`)
- HPCC wiki (`https://wiki.hpcc.msu.edu/x/4ACE`)

Buyin account

- Faculty can purchase nodes and get a buyin account for all group members.
- Buyin users submit jobs under their buyin account (by default). Their jobs will start running on the buyin nodes within 4 hours.
- Non-buyin users submit jobs under the *general* account (by default). Their jobs have full access to all non-buyin nodes; if the job requests a wall time less than 4 hours, it can also run on buyin nodes.

HPCC job policy

 <https://wiki.hpcc.msu.edu/x/SgYzAQ>

Limits per user:

- 1 million CPU hours per year (only for general account users)
- Maximum wall time for a job: 7 days
- Maximum number of jobs in queue: 1000
- Maximum CPU cores at one time: 520

Example 1: single node, single core

- The simplest situation, also the primary type of user applications.
- You only need to specify resource of **memory** and **wall time**.
- You can ask for email notifications if needed.

Script `first.sbatch`

```
#!/bin/bash

# Job name:
#SBATCH --job-name=first
#
# Memory per node:
#SBATCH --mem=20M
#
# Wall time (e.g. "minutes", "hours:minutes:seconds", "days-hours", "days-hours:minutes"):
#SBATCH --time=5
#
# Mail type:
#SBATCH --mail-type=ALL
#
# Mail user:
#SBATCH --mail-user=yournetid@msu.edu
#
# Standard out and error:
#SBATCH --output=%x-%j.SLURMout

echo "JobID: $SLURM_JOB_ID"
echo "Running on node: `hostname`"

module purge
module load GCC/8.3.0 OpenMPI/3.1.4 R/4.0.2

Rscript lincoef.R > lincoef.Rout
```

⚠ SLURM stops reading directives at the first executable (i.e. non-blank, and doesn't begin with #) line.

Submitting the job

```
sbatch first.sbatch
```

Example 2: single node, multiple cores (`tophat` example)

1. Passing shell environment variables to your script (e.g., to customize your job name and output file name);
2. Embedding SLURM environment variables in your own command;
3. Using `--constraint` to run jobs only on a certain type of nodes.

Script `tophat.sbatch` (part 1: directive lines)

```
#!/bin/bash

# Number of nodes needed:
#SBATCH --nodes=1
#
# Tasks per node:
#SBATCH --ntasks-per-node=1
#
# Processors per task:
#SBATCH --cpus-per-task=6
#
# Memory per node:
#SBATCH --mem=25G
#
# Wall time (e.g. "minutes", "hours:minutes:seconds", "days-hours", "days-hours:minutes"):
#SBATCH --time=3:00:00
#
# Mail type:
#SBATCH --mail-type=ALL
#
# Mail user:
#SBATCH --mail-user=yournetid@msu.edu
```

Script `tophat.sbatch` (part 2: your commands)

```
echo "SLURM_NTASKS: $SLURM_NTASKS"
echo "SLURM_CPUS_ON_NODE: $SLURM_CPUS_ON_NODE"

module purge
module load GCC/5.4.0-2.26 OpenMPI/1.10.3
module load TopHat/2.1.1
module load Bowtie2/2.3.2
module load SAMtools/1.5

# The shell variable "sample" is provided by sbatch command line arg passing
sampleFastq1=${sample}_1.fastq
sampleFastq2=${sample}_2.fastq

if [ ! -d $sample/map ]; then
    mkdir -p $sample/map
else
    if [ -e $sample/map/tophat.log ]; then
        rm $sample/map/tophat.log
    fi
fi

# Run Tophat
tophat2 -p $SLURM_CPUS_ON_NODE -o $sample/map \
    --transcriptome-index=/mnt/research/common-data/Bio/Ensembl_GRCh38_GTF/GRCh38 \
    --read-realign-edit-dist 0 \
    -g 1 -x 1 -m 2 -r 80 \
    --library-type fr-unstranded \
    /mnt/research/common-data/Bio/Ensembl_GRCh38_unmasked_ref/GRCh38 $sampleFastq1 $sampleFastq2 \
    > $sample/map/tophat.log 2>&1
```

Submitting the job

```
n="tophat" # job name

# - Each job contains one sample's analysis work.
# - Using a bash loop, we can submit multiple samples' jobs conveniently, and let them run in parallel.
# - Job name, output file name and sample names are easily passed through command line.
# - Because "bowtie" can only run on intel16 (or higher) nodes, we need to put a constraint.

for S in ERR315325sub ERR315326sub ERR315382sub ERR315424sub
do
    sbatch --job-name=$n --output=$n.$S.SLURMout --export=sample=$S --constraint="[intel16|intel18]" \
        tophat.sbatch
done
```

Example 3: multiple nodes (e.g., an MPI job)

1. When running an MPI job across multiple nodes, memory request in SLURM should be on a per CPU basis (`--mem-per-cpu`);
2. Each MPI rank (or process) is a task and so we need to specify `ntasks` (i.e., number of tasks).

SLURM will determine how many nodes and tasks per node are needed.

3. In general, need to use `mpirun -n $SLURM_NTASKS` to launch the MPI program within SLURM script.

(⚠ there is an **exception with Rmpi** which requires the use of `mpirun -n 1`)

mothur MPI job script mothur.sbatch

```
#!/bin/bash

# Job name:
#SBATCH --job-name=mothur_test
#
# Number of MPI tasks needed for use case:
#SBATCH --ntasks=8
#
# Processors per task:
#SBATCH --cpus-per-task=1
#
# Memory:
#SBATCH --mem-per-cpu=100M
#
# Wall clock limit:
#SBATCH --time=30
#
# Standard out and error:
#SBATCH --output=%x-%j.SLURMout

module purge
module load icc/2017.1.132-GCC-6.3.0-2.27 impi/2017.1.132 Mothur/1.40.3-Python-2.7.13 # Mothur MPI version

mpirun -n $SLURM_NTASKS mothur batch.m
```

⚠ make sure the mothur commands as put in `batch.m` are set with a matching number of processors.

Submitting the job

```
sbatch --constraint="[intel16|intel18]" mothur.sbatch
```

Rmpi job script Rmpi.sbatch (skip it if you're not an R user)

```
#!/bin/bash

#SBATCH --job-name=Rmpi
#SBATCH --ntasks=30
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=500M
#SBATCH --time=30
#SBATCH --output=%x-%j.SLURMout

echo "SLURM_NTASKS: $SLURM_NTASKS"

# Load R v3.5.1
module purge
module load GCC/7.3.0-2.30 OpenMPI/3.1.1 R/3.5.1-X11-20180604

# Suppress warnings about forks and missing CUDA libraries
export OMPI_MCA_mpi_warn_on_fork=0
export OMPI_MCA_mpi_cuda_support=0

mpirun -n 1 Rscript monte-carlo-pi.R > monte-carlo-pi.Rout
```

Submitting the job:

```
sbatch Rmpi.sbatch
```

Rmpi : why setting `mpirun -n 1` ? (skip it if you're not an R user)

- In many distributed applications `mpirun` starts `N` processes, and they exchange messages.
- In the case of `Rmpi`, the *manager* starts all the *workers* itself, so we do not want `mpirun` to start any additional copies of R. In other words, if the program itself can spawn workers, we won't need `mpirun` to do that again.
- In the case of other R MPI packages, such as `pbdsMPI`, use `mpirun -n $SLURM_NTASKS Rscript xxx.R` as in the mothur example.

Always consult the manual.

Advanced node requirements

Specifying nodes to run your jobs

```
--nodelist=<node name list>
```

Request a specific list of hosts. The job will contain all of these hosts and possibly additional hosts as needed to satisfy resource requirements. The list may be specified as a comma-separated list of hosts, a range of hosts (host[1-5,7,...] for example), or a filename.

Excluding nodes

```
--exclude=<node name list>
```

Explicitly exclude certain nodes from the resources granted to the job.

Check buyin nodes

⚠ You need to load `powertools` in order to use this command.

```
module load powertools
buyin_status -a genome_lab # buyin group name is "genome_lab" (hypothetical)
```

Buyin: genome_lab						
JOBID	STATE	USER	CPUS	PRIORITY	TIME_LIMIT	START_TIME
xxxxx	RUNNING	user1	1	20082	3:59:00	2018-09-13T13:46:48
Partition: genome_lab-16						
lac-001 (mixed)						
JOBID	ACCOUNT	USER	CPUS	TIME	TIME_LEFT	
xxxxx	general	user2	1	6:28	23:32	
xxxxx	general	user2	1	6:28	23:32	
lac-002 (allocated)						
JOBID	ACCOUNT	USER	CPUS	TIME	TIME_LEFT	
xxxxx	general	user3	1	10:00	21:32	
xxxxx	genome_lab	user4	1	8:20	21:32	

- `genome_lab` has two buyin nodes (`lac-001` and `lac-002`).
- Jobs under `Buyin: genome_lab` come from group members.
- Jobs under `Partition: genome_lab-16` come from members and non-members (non-members jobs must have a walltime less than 4 hrs).

Check a specific job

```
sacct -S 2019-01-01 -D --units=G \  
  --format="state%15,Account%15,JobID%20,JobName%15,TimeLimit,Elapsed,TotalCPU%12,NCPUS,\  
  NodeList%25,NNodes,ReqMem,MaxRSS,Submit,Start,End" -j 21581
```

Check jobs during a particular period of time

```
sacct -D -X --units=G -S 2019-01-01 \  
      --format="state%25,Account%18,JobID%20,TimeLimit,Elapsed,NCPUS,NNodes,ReqMem,Submit,Start,End"
```

- `-S` : starting after a specified time
- `-E` : ending before a specified time (default: midnight of the current day)

Show jobs in the queue

Your own jobs:

```
queue -u $USER \  
-0 state,jobid,username,submittime,starttime,timelimit,maxnodes,maxcpus,minmemory,account,reason
```

All users jobs:

```
queue -0 \  
state,prioritylong,jobid,username,submittime,starttime,timelimit,maxnodes,maxcpus,minmemory,account,reason
```

Check node availability

⚠ You need to load `powertools` in order to use this command.

```
module load powertools
active_nodes
```

```
*****
Showing node availability:

(1) Memory values are in GB.
(2) Nodes in "down" / "drained" / "reserved" states are not displayed.
(3) State "alloc" means all CPUs on the node have been allocated;
    "mix" means some CPUs idle and other CPUs allocated.
*****

HOSTNAMES  CPUS   CPU_LOAD  MEMORY  FREE_MEM  STATE  AVAIL_FEATURES  NODETYPE
nvl-000    40     12        367     221       mix    skl,gbe,intel18,ib,edr18,v100,gpgpu  Buyin
nvl-001    40     12        367     302       mix    skl,gbe,intel18,ib,edr18,v100,gpgpu  Buyin
nvl-002    40     27        367     256       alloc  skl,gbe,intel18,ib,edr18,v100,gpgpu  Non-buyin
nvl-003    40     23        367     322       mix    skl,gbe,intel18,ib,edr18,v100,gpgpu  Non-buyin
nvl-005    40     7         367     283       mix    skl,gbe,intel18,ib,edr18,v100,gpgpu  Non-buyin
nvl-006    40     21        367     134       mix    skl,gbe,intel18,ib,edr18,v100,gpgpu  Non-buyin
skl-000    40     43        84      68        mix    skl,gbe,intel18,ib,edr18             Buyin
skl-001    40     10        84      74        mix    skl,gbe,intel18,ib,edr18             Buyin
skl-002    40     10        84      74        mix    skl,gbe,intel18,ib,edr18             Buyin
...

```

Check specific nodes

```
# Show a single node  
scontrol show node=lac-251 -a  
  
# Show consecutive nodes with one-liner view  
scontrol show node=lac-[295-299] -a --oneliner | less -S
```