Introduction to batch computing using SLURM

Riccardo Murri <riccardo.murri@uzh.ch>

2018-04-17

What is a batch-queueing system?

A batch-queuing system is a way to execute computational jobs asynchronously.

You submit a script to be processed to a central resource scheduler, and the scheduler executes the script when enough resources (CPUs, memory, disk space, etc.) are available.

Note: This also means that you cannot *interactively* type input to the script! Batch computing is mostly intended for *non-interactive* tasks.

What is a batch-queueing system?

> *A batch-queuing system is a way to execute computational jobs asynchronously.*

You submit a script to be processed to a central resource scheduler, and the scheduler executes the script when enough resources (CPUs, memory, disk space, etc.) are available.

Note: This also means that you cannot *interactively* type input to the script! Batch computing is mostly intended for *non-interactive* tasks.

What is a batch-queueing system?

> *A batch-queuing system is a way to execute computational jobs asynchronously.*

You submit a script to be processed to a central resource scheduler, and the scheduler executes the script when enough resources (CPUs, memory, disk space, etc.) are available.

Note: This also means that you cannot *interactively* type input to the script! Batch computing is mostly intended for *non-interactive* tasks.

What is SLURM?

SLURM is one of many batch-queueing systems available for GNU/Linux clusters. It is the system available on `cluster.pelkmanslab.org`.

The SLURM batch-queuing system can distribute job execution across a cluster of independent computing nodes so that many jobs can be executed at the same time.

Reference: http://slurm.schedmd.com/

SLURM commands

How do I interact with SLURM?

SLURM provides several commands to control job submission and status:

- `sbatch` – enqueue a *shell script* for execution
- `squeue` – display information about running or pending jobs
- `sacct` – display information about completed jobs
- `srun` – run command interactively

All these commands provide a `man` page full of information. (E.g., run `man sacct`)

sbatch

You can submit a *shell script* for processing by SLURM using the `sbatch` command like this:

```
sbatch my_script.sh
```

Using the `sbatch` command, by default your job will run on 1 CPU and using the entire memory of a compute node (32GB).

Options to `sbatch` are available to request more CPUs or a different memory slice.

Example: run MATLAB command

The following script can be used to execute MATLAB commands in a SLURM job:

```
#!  /bin/sh
#
# Run MATLAB command(s) non-interactively.
# Note:  if you want to run a MATLAB file,
# strip away the '.m' file name ending.
#
exec matlab -nodesktop -nodisplay -r "$@"
```

You can submit this MATLAB job to SLURM by running:

```
# note:  it's 'hello' not 'hello.m'
sbatch run_matlab_cmd.sh hello
```

squeue

Once you have submitted a job, you can monitor its status with `squeue`:

```
$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
  502      main run_matl   rmurri  R       0:03      1 pelkmanslab-slurm-worker-001
```

Note: once the job is finished, it won't be listed by `squeue` anymore.

sacct

You can list already-finished jobs with the `sacct` command:

```
$ sacct
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
502          run_matla+       main       root          1  COMPLETED      0:0
502.batch         batch                  root          1  COMPLETED      0:0
```

Note: By default, `sacct` lists only jobs finished since last midnight (and until now). Use options `--starttime` and `--endtime` to change this window.

Where have all the outputs gone?

Standard output *and* standard error output of a job are (by default) collected in a file named `slurm-JJJ.out`:

```
$ cat slurm-502.out

                            < M A T L A B (R) >
                  Copyright 1984-2017 The MathWorks, Inc.
                   R2017b (9.3.0.713579) 64-bit (glnxa64)
                              September 14, 2017


To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

Hello, SLURM!
```

This can be changed using options `-o` and `-e` to `sbatch`.

(Common) Options to `sbatch`

You can specify options to `sbatch` directly *in the job script*, as special comments introduced by the `#SBATCH` prefix:

```
## use 1 CPU
#SBATCH -c 1

## run for (max) 8 hours
#SBATCH --time=8:00:00

## use (max) 3500 MB of memory per CPU
#SBATCH --mem-per-cpu=3500m

## write both output and errors to file `run_matlab.NNN.log`
#SBATCH -o run_matlab.%j.log
#SBATCH -e run_matlab.%j.log
```

(Common) Options to `sbatch`

You can also specify options to `sbatch` on the command line:

```
$ sbatch --time=8:00:00 \
         --mem-per-cpu=3500m \
         run_matlab_cmd.sh hello
```

If the same option is given *both* in the script *and* on the command line, the command-line wins.

squeue

You can check the state of (anyone's) jobs with the `squeue` command:

```
# view all jobs
squeue

# view only jobs of $USER
squeue -u $USER

# view only running jobs (af any user)
squeue --state RUNNING
```

Changing the default output of squeue

Environmental variable `SQUEUE_FORMAT2` sets the columns that are displayed in the table produced by command `squeue`:

```
$ export SQUEUE_FORMAT2="jobid:6,username:12,name,state,reason,timeused:12,timeleft:12"
$ squeue
JOBID USER        NAME                STATE       REASON              TIME        TIME_LEFT
504   rmurri      sleep.sh            RUNNING     None                0:43        UNLIMITED
```

Read `man squeue` to find out all the possible column names.

If you don't like the default `squeue` columns, you can (and should!) set a new value for `SQUEUE_FORMAT2` in your `.bashrc` file.

Changing the default output of `sacct`

Environmental variable `SACCT_FORMAT` sets the columns that are displayed in the table produced by command `sacct`:

```
$ export SACCT_FORMAT="jobid,user,state,exitcode,start,elapsed"
$ sacct
       JobID      User      State ExitCode               Start    Elapsed
------------ --------- ---------- -------- ------------------- ----------
502             rmurri  COMPLETED      0:0 2018-04-16T14:27:33   00:00:13
502.batch               COMPLETED      0:0 2018-04-16T14:27:33   00:00:13
```

Use `sacct --helpformat` to list all possible column names.

If you don't like the default `sacct` columns, you can (and should!) set a new value for `SACCT_FORMAT` in your `.bashrc` file.

Interactive sessions

Use command `srun --pty` to run a job interactively.

For instance, to start a shell on a compute node:

```
$ srun -pty bash -login
```

Note: The `srun` command will block until an execution slot is available.

Dealing with multiple software versions

The Bad News

> *There is no generic way of installing (and easily switching) multiple versions of the same software or software stack.*

The easy-to-use systems are language- or software-specific (e.g., Python).

The generic systems revolve around controlling environmental variables (but have a lot of caveats and edge cases).

Solution for Python: virtualenv

Python has “virtual environments”, which are just directories containing a copy of Python and libraries.

Once you *“activate”* a virtualenv, every time you install a Python library (`pip install`), it is installed in the virtualenv.

You can delete a virtualenv by simply removing its directory:

```
$ rm -r my_python_virtualenv
```

Using Python's virtualenv

- **Create a virtualenv** (once only)

  ```
  $ python -m virtualenv /path/to/venv/
  ```

  Note:

  - The virtualenv directory *must not* exist!
  - Any filesystem path is fine, but choose one on a shared filesystem (e.g. your home directory) if you want to use it in jobs across the cluster.

- **Enter ("activate") a virtualenv** (in every new shell)

  ```
  $ source /path/to/venv/bin/activate
  ```

- **Exit a virtualenv**

  ```
  $ deactivate
  ```

Using virtualenvs in SLURM jobs

Commands in a job script are no different from the commands you type at the shell prompt, so just activate the virtualenv.

```
# '.' is an alias for 'source' that works always
venv="$HOME/path/to/virtualenv"
. "$venv/bin/activate"

# 'exec' must be the very last statement
exec python my_script.py
```

The generic solution: *environment modules*

The `module` command is the standard solution on HPC clusters for managing multiple versions of the same software.

In essence, just manipulates the shell environment, adding or removing variables, or changing the value of the existing ones (e.g., add prepend directories to `PATH`). All the changes are listed in a single (Lua-syntax) file, and are performed (or undone) at the same time.

Listing available modules

You can list available module files with the command `module avail` (can be abbreviated to `ml av`):

```
$ ml av

--------------------------- /opt/modulefiles ----------------------------
   MATLAB/R2013b    MATLAB/R2017b (D)

-------- /opt/spack/share/spack/modules/linux-ubuntu16.04-x86_64 --------
   intel-parallel-studio-professional.2018.0-gcc-5.4.0-qx4ndcf

---------------------- /opt/easybuild/modules/all ------------------------
   EasyBuild/3.5.2    EasyBuild/3.5.3 (D)

------------------- /opt/lmod/lmod/modulefiles/Core ---------------------
   lmod/7.0    settarg/7.0

  Where:
   D:  Default Module
    [...]
```

Note that each module name consists of a “software name” and a “version”, separated by a slash “/”.

Loading modules

The command `module load` (short: `ml`) performs all the changes to the environment described in a module file, and "activates" a specific version of software:

```
  $ ml MATLAB/R2013b

  $ matlab -nodisplay -nodesktop -r version
  [...]
ans =

8.2.0.701 (R2013b)
```

Loading another version of the same module unloads the first one automatically:

```
  $ ml MATLAB/R2017b

The following have been reloaded with a version change:
  1) MATLAB/R2013b => MATLAB/R2017b
```

Example: MATLAB module file

Command `module show` displays the contents of a module file:

```
$ ml show MATLAB/R2013b
-------------------------------------------------------------------------------
   /opt/modulefiles/MATLAB/R2013b.lua:
-------------------------------------------------------------------------------
help([[
  This module loads MATLAB path and environment variables.
]])
prepend_path("PATH","/opt/MATLAB/R2013b/bin")
```

Here, `MATLAB/R2013b` is the module for software `MATLAB`, version `R2013b`, stored in file `MATLAB/R2013b.lua`.

Complete list of functions that can be used in a Lua module file: http://lmod.readthedocs.io/en/latest/050_lua_modulefiles.html

Where to put my own module files?

The command `module use` adds a directory to the search path for module files.

The usual choice is to store one's own module files into `$HOME/modulefiles`.

You can add the `module use` command to your `.bashrc` file to automatically add a module directory each time you start a new shell.

How to use `module` in shell scripts

The `module` command is loaded by file `/etc/profile.d/modules.sh` which you need to *source*:

```
#!  /bin/bash

# load the 'module' command
. "/etc/profile.d/modules.sh"

# load an older version of MATLAB
module load MATLAB/R2013b

# run it
exec matlab -nodesktop -nodisplay -r version
```

Job arrays

Job arrays

A *job array* is a $1D$ collection of jobs; each job is distinguished by its *index* in the array (an integer number).

Job arrays display as a single row in `squeue` output, but effectively many jobs will be run.

Job arrays are created using the `--array` option to `sbatch`:

```
# submit an array of 12 jobs
sbatch --array 0-11 array.sh

# submit 100 jobs, but only allow 10
# to run at the same time
sbatch --array '0-99%10' array.sh
```

Example: Job array script

A job array script is like any job script, but you should use the `$SLURM_ARRAY_TASK_ID` environment variable to detect the task to run.

```
# note:  `case ...  in` uses *string* comparison!
case "$SLURM_ARRAY_TASK_ID" in

    "1")
        param1=1.5
        param2=3.4
        exec matlab -r simul $param1 $param2
        ;;

    "2")
      # ...

esac
```

The End

Thank you!

Any questions?

Thank you!

Any questions?