Introduction to batch computing on the Flux cluster: Lmod modules

07 Oct 2016

1. Software modules
   - Software modules enable you to control which software packages are available to use to prevent conflicts and control versions. We use a module program called Lmod, from the TACC
   - How to find software that is installed

     ```
     $ module keyword statistics
     ```

   - To get some more information about a package

     ```
     $ module spider stata
     ```

   Note how it displays some information and gives you pointers to near misses. Not a perfect system, though

   ```
   $ module spider R
   ```

   that worked pretty well, but what about

   ```
   $ module spider openmpi
   ```

   scroll down until you find `openmpi/1.10.2/gcc` and there you will see the versions. Lmod considers `openmpi/1.10.2/gcc` to be the package name and the `4.8.5`, `4.9.3`, and `5.3.0` the version numbers.

   - To make software available, you *load* it.

     ```
     $ module load R
     ```

   to see what is loaded, use

   ```
   $ module list
   ```

   to unload a loaded module, use

   ```
   $ module unload R
   ```

   to clear out all the loaded modules, use

   ```
   $ module purge
   ```

   - `module` commands need to be run once after logging in and before submitting a job to the queue. That insures that the settings are communicated to all the processes on all the nodes (using the `-V` PBS flag).

- If you find you need the same module(s) all the time, you can create module *sets*. These are exactly what they sound like: A set of modules that can be treated as a unit. I recommend that you create sets after a fresh login and after purging loaded modules.

  ```
  $ module purge
  $ module load R
  $ module load python-anaconda2
  $ module list
  ```

Now you have a couple of modules loaded, save this as `stats` using

```
$ module save stats
```

To load a saved set, use

```
$ module restore set_name
```

To see the names of the save sets, use

```
$ module savelist
```

and to see what is in a set use

```
$ module describe set_name
```

Finally, there is a special set called `default` that will load automatically when you login. You create is the same way as any other set, but either name it `default` explicitly, or leave off the name.

Sets are saved as files in `~/.lmod.d`, and if you want to delete one, use the `rm` command.

- Make another set called `build-intel` that contains

  ```
  intel/16.0.3 openmpi/1.6.5/intel/16.0.3
  ```

Now, purge your modules, restore the `stats` set. List your modules. Now restore your `build-intel` set. What are your loaded modules? So, restoring is not the same as loading a list; it's a list of all the modules that will be loaded after that state is restored.

- Modules can have dependencies

  ```
  $ module spider gromacs
  ```

We want `gromacs/5.1.2/openmpi/1.10.2/gcc/4.9.3`. What does that name tell you? That's parsed as software/version pairs, where each package depends on the software/version(s) to its right. So, gromacs depends on openmpi which depends on gcc. Try loading that

```
$ module load gromacs/5.1.2/openmpi/1.10.2/gcc/4.9.3
```

Lo and behold, there are some other prerequisites, too. You can copy and paste the dependencies onto a module load, and that's a set worth saving, by gum!

- Modules can have conflicts

  ```
  $ module purge
  $ module restore stats
  $ module load python-epd
  ```

2. Now we have the pieces to run software that is already runnable, e.g., things that are installed on Flux and don't require anything additional to run, like R, Stata, Matlab.
   - Example of developing a working script. Supose we are embarking on a study of plant phylogeny. We have some data already stored in an R library. As the study progresses, the same statistics will need to be run as each round of data is added. So, first we determine, using a small sample, on the login node which commands are needed and to check that we have syntax correct.

     ```
     $ module load R/3.1.1
     $ R

     > library(datasets)
     > data(iris)
     > summary(iris)
     ```

   - Now that we have that working, we need to put that into a script and check that the batch version works. Enter those commands into `iris.R`

     ```
     $ mkdir R-examples
     $ cd R-examples
     $ R CMD BATCH --no-restore --no-save iris.R
     ```

   - Next, we need to make a PBS script to run the batch job. Copy your `template.pbs` to `iris.pbs`, fill in the necessary options to run the R script and submit it.

Optional section on compiling software

1. Compiling software
    - Serial or shared memory software – one physical machine
    - Use the compiler directly

      ```
      $ cp -r /scratch/data/workshops/hpc101/code-examples .
      $ gcc -o hello hello.c
      $ ./hello
      ```

    - To change compiler

      ```
      $ module load intel
      $ gcc -o hello hello.c
      $ ./hello
      ```

    - For this simple example, `gcc` and `intel-comp` can coexist, however it is good practice to unload the unused compiler if you can.
    - Software you compile yourself is run the same way as commercial software
    - Parallel or MPI-based software – more than one physical machine
    - Uses compiler scripts instead of using the compiler directly
    - Compiler and MPI version must match
        - Check with `module av openmpi`, which gives yet another view of the choice of modules; which modules contain `openmpi` somewhere in the name.

          ```
          $ mpicc -o hello mpi-hello.c
          $ mpirun -np 4 hello
          ```

    - You must load the correct (matching) openmpi to run a program compiled with it
    - When run from inside a batch job, you do not need `-np N` unless you use fewer processors than requested by the PBS script.
    - To change compilers, you have to change both the compiler and the openmpi

      ```
      $ module purge
      $ module load gcc/4.9.3 openmpi/1.10.2/gcc/4.9.3
      $ mpicc -o hello mpi-hello.c
      $ mpirun -np 4 hello
      ```