NBIC TechTrack PBS Tutorial

by Marcel Kempenaar,
NBIC Bioinformatics Research Support group*,
University Medical Center Groningen

*Visit our webpage at: `http://www.nbic.nl/support/brs`

NBIC PBS Tutorial

This document provides an introduction to clusters and the PBS software. Users with some experience into using a cluster can safely skip these sections and use it as a reference when doing the exercises.

Tutorial Cluster Information [todo]

This tutorial consists of 10 exercises that can all be performed on the cluster that we've setup for this tutorial. To connect to this cluster use the following command from your terminal (or use `Putty` when using a Windows operating system):

```
ssh <user>@<server>
```

Request your temporary username and password to login from one of the teachers. In your home directory there is a folder named `example_scripts` which contain the scripts referred to later on.

Cluster Introduction

Of the many types of existing clusters, we will be using a '*Compute Cluster*' for this tutorial[†]. The concept of such a cluster often is to use commercial off-the-shelf (COTS) computers (nodes) and combine large amounts of these systems into a networked compute cluster. The nodes are linked using fast network connections (*e.g.* InfiniBand or similar fiber-optic connections) and have a shared filesystem.

The end-user, who wants to run his computational intensive tasks on such a cluster, can connect to one of these node (most often a dedicated login-node, also referred to as the user interface*ui*- node) and start the desired tasks on the available worker nodes.

Applications of a Cluster

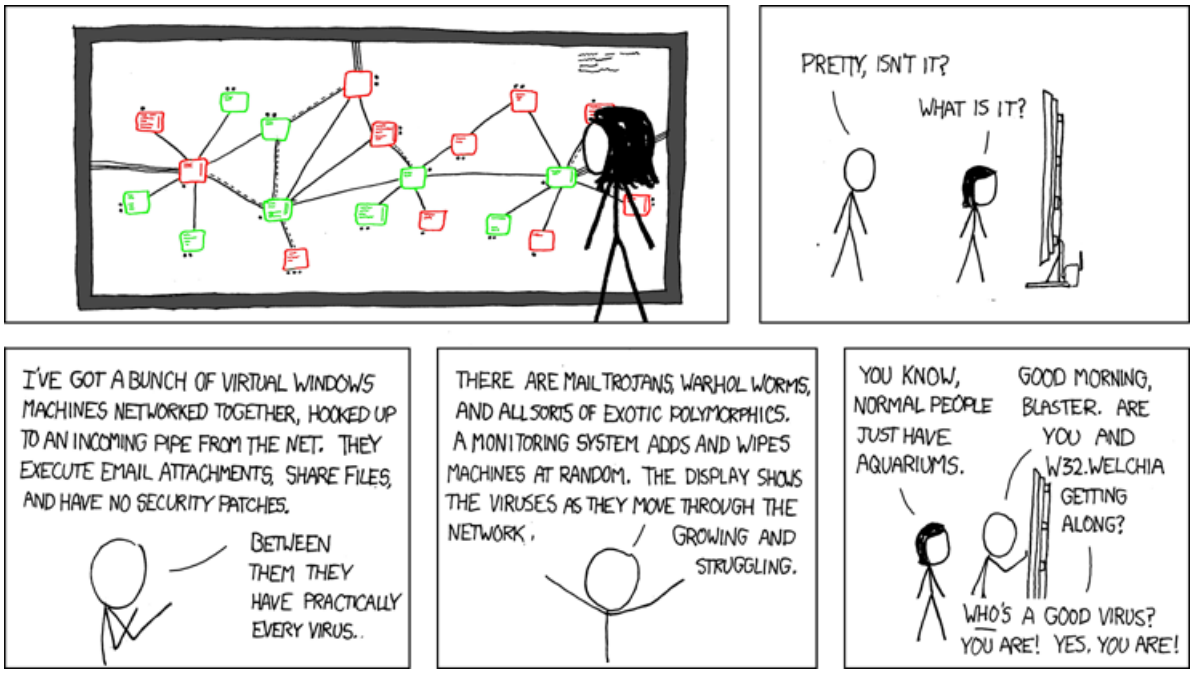
Clusters have many purposes and are an affordable (most of the time) alternative to a supercomputer while capable of offering similar performance.

PBS Introduction

The Portable Batch System (PBS) is the software that performs the important task of job scheduling on a cluster. A scheduler prioritizes, starts and monitors the jobs which is needed since most clusters are shared by multiple users. A scheduler on the cluster basically the same as the scheduler on a normal computer (such as `CRON` for Unix and the `Task Scheduler` for Windows) with the difference that a cluster scheduler monitors many more resources (nodes) and jobs (up to thousands of active/queued jobs simultaneously).

When starting a job on the cluster, the scheduler will check if there are free resources which can be used for that job and submits it to the appropriate nodes for execution or places it in a queue if there are no available resources.

[†]See the Wikipedia page on clusters for examples of other cluster types: `http://en.wikipedia.org/wiki/Computer_cluster`



One of the advantages of PBS is scaling; a job that runs on a few nodes can easily be run on a cluster (such as `LISA`[‡]) containing thousands of nodes.

PBS Script

In order to run a job on a cluster it is necessary to construct a *job script* first. This script contains the commands that you want to run, in essence this can be any standard shell script (`.sh` file) to which we will add a header with some commands to instruct the scheduler. Following is a short description of these commands (also called *PBS directives*) with examples for each. During the exercises we will see some more advanced features of these PBS scripts. All PBS commands are preceded by a '`#`' which makes it a comment-line in the shell script. This means that a PBS script is executable in a shell which makes it easy for testing. It is advised to always test your applications (more on testing later on) to monitor runtimes and memory usage, since these are required to be given in the PBS script as shown below.

Job Title: While optional, it is advised to give the job an unique name to be able to recognize it when running multiple jobs. The following command sets the title to `MyFirstPBSJob`:

```
#PBS -N MyFirstPBSJob
```

Nodes and Cores: The user is required to set the number of nodes and cores per node that are required for the job. This is of course depending on the cluster structure, not all nodes are multi–core for instance. If your job supports

[‡]The `LISA` grid at SARA: `https://subtrac.sara.nl/userdoc/wiki/lisa64/description`

multithreading you can use the following configuration line to specify 1 node for the job with 4 cores:

```
#PBS -l nodes=1:ppn=4
```

Walltime: the time entered here will be the maximum time the job will run. This is not the same as the CPU-time, but the total time spend for all included actions (*as shown by the clock on the wall*). The schedular will monitor the time for each job and kills it when this exceeds the specified walltime. Although this might sound difficult since it is sometimes hard to estimate the runtime for a job, it is essential for the scheduler to do the actual scheduling. When entering a long walltime, the job might get queued for some time since there might not be space to fit in your job. With a smaller walltime the chances are that it might fit in sooner, but you risk the job being stopped if your estimations are off. Also, choosing the walltime affects the queues that can be used (see Section **??**). To set the maximum walltime for a particular job (*e.g.* 12 hours), use the following line in the PBS job script header:

```
#PBS -l walltime=12:00:00
```

Memory: This sets the maximum amount of memory that your job will (be able to) use. Depending on the cluster settings, using more memory might result in the job being stopped by the schedular. Most often this is not as strict as when exceeding the walltime for instance. As with the walltime, setting this number to high can result in long queue times as the scheduler waits for available space. To set a maximum memory usage of 4GB, use the following line:

```
#PBS -l mem=4GB
```

Logfiles: All jobs create log files by default that can be used to check for errors and to review output of the tool. The two types of log files can be defined by:

```
#PBS -e MyFirstPBSJob.elog
#PBS -o MyFirstPBSJob.olog
```

where the `-e` log file shows all errors that occurred and the `-o` log file shows the output that is normally printed on screen. If you don't include these lines in your script, the log files are named like: `<job_name>.(e/o)<job_id>`.

There are some other PBS directives that can be used but those will not be further explained or used in this tutorial. For example it is possible to receive e-mail notifications when a job starts or ends ('`#PBS -m b`' and '`#PBS -m e`' respectively). The rest of the script can consist of basic commands that would run on the terminal and those constitute the functionality of the job.

Remember that the directory in which the job is run is the users home directory. Therefore you should always include the paths to files or programs that are used in your job. You can change the working directory by simply putting a '`cd [dir]`' command in your script, like you would do in the terminal.

Another note, the cluster will run your script by default in the same terminal that you use on the head node (for instance C-shell (`csh`)). You can change

the shell to another type by putting this on the first line of your job file as with regular shell scripts. To use the `bash` shell, place the following line at the start of your file:

```
#!/bin/bash
```

Queues

Most clusters have different queues for different purposes and each with different properties. The clusters resources are divided amongst these queues and access can be controlled by the admin. Often parameters such as the maximum wall time or the maximum number of simultaneous jobs are setup differently. Depending on the properties of your job you can select a queue that best fits your job and install your job in that queue. To see a list of all queues available on the cluster, use the `qstat`[§] command:

```
qstat -Q
```

Example output[¶]:

```
Queue              Max   Tot   Ena   Str   Que   Run   Hld   Wat   Trn   Ext T
----------------   ---   ---   ---   ---   ---   ---   ---   ---   ---   --- -
quadslong            0    74   yes   yes    66     6     0     2     0     0 E
quads                0   100   yes   yes   100     0     0     0     0     0 E
nodeslong            0   966   yes   yes   527   438     1     0     0     0 E
nodes                0   359   yes   yes   274    72     0    13     0     0 E
short                0     1   yes   yes     1     0     0     0     0     0 E
```

Next to the names of the queues, the list shows the number of active (running) jobs, queued jobs, and jobs on hold. An alternative view, using:

```
qstat -q
```

Example output:

```
Queue            Memory CPU Time Walltime Node  Run Que Lm  State
---------------- ------ -------- -------- ----  --- --- --  -----
quadslong          --      --    240:00:0   --    6  68 --   E R
quads              --      --    24:00:00   --    0 100 --   E R
nodeslong          --      --    240:00:0   --  438 528 --   E R
nodes              --      --    24:00:00   --   72 287 --   E R
short              --      --    00:30:00   --    0   1 --   E R
                                               ----- -----
                                               516 984
```

shows important information about the maximum walltime of the queues. Even more detailed information can be obtained with:

```
qstat -Qf
```

[§] Use the `man qstat` command to get help for this command. All other commands from this tutorial also have man-pages that can be very helpful.

[¶] All example outputs shown are from the Millipede cluster at the University of Groningen: `http://www.rug.nl/cit/hpcv/faciliteiten/hpccluster`

Submitting a job on the cluster

Once you have a shell script with added PBS header lines and selected a queue to run your job on, you can submit this job to the cluster using the `qsub` command as follows:

```
qsub -q -V <queue> <job_script.sh>
```

All the commands (the '*PBS directives*') can also be entered on the command line with `qsub`, however it is advised to place them in a job script. Executing a shell script called `MyFirstPBSJob.sh` on the cluster with the settings as shown in Section **??**, would be done with:

```
qsub -V -N MyFirstPBSJob -l walltime=12:00:00 -l mem=4GB
     -l nodes=1:ppn=4 -q nodes MyFirstPBSJob.sh
```

Environment Variables

You may use environmental variables in your script. For example, the path to your home directory is contained in the variable `$HOME`. Instead of hardcoding the path to your home, you may substitute the absolute path to your home directory with the variable `$HOME`. You need to use the `-V` argument to `qsub` to specify that you are using environment variables. Even when not using them, as a best practice, always use the `-V` option.

Viewing the status of your job(s)

Once a job is submitted on the cluster its status can be viewed with the `qstat` command. If executed without any arguments, it shows all the jobs from all users (depending on cluster settings). To view all of your own jobs that are currently running or queued, use:

```
qstat -u <username>
```

Viewing the status of a single job is also possible when you know the `job ID`, which was printed on the screen when submitting the job. Using this ID (for instance, `1523059`) we can view its status using:

```
qstat 1523059
```

Extra details can be seen when using the `-f` flag, including the actual used memory and the node the job is distributed to or running on.

Another command that is useful to get an overview of all your jobs is the following:

```
showq -u <username>
```

Removing a submitted job from the queue

There is a simple command to stop a job that has already been submitted to the cluster. Using the job ID (for instance, `1523059`), we can stop this particular job with:

```
qdel 1523059
```

When we want to stop all of our jobs, the command would be:

```
qdel all
```

This command will try to stop all jobs on the cluster, not just your own, which most likely will generate a list of errors (`Deletion Error (Unauthorized Request)`) that can safely be ignored. It has however stopped all the jobs that were running for the user that executed the command.

Exercises

Exercise 1; Submitting a job

Todo: create simple example script that has a runtime of approx. 1 min (or use the keep_completed flag (`http://www.clusterresources.com/torquedocs/2.5completedjobs.shtml`))

Take a look at the first example script (`cat` prints the contents of a file in the terminal):

```
cd example_dir
cat test1.sh
```

As you can see this script contains PBS directives in its header followed by a few standard Unix commands. As mentioned before, a PBS–script can be executed in the terminal like a regular shell script. Execute the script and note its output.

```
./test1.sh
```

Now use the `qsub` command and execute the same script (using the default queue).

```
qsub example01.sh
```

The output should look something like:

```
7135.gbcetud.ewi.tudelft.nl
```

Exercise 2; Selecting a queue and job monitoring

Before submitting your cluster job again, take a look at the available queues and select one of the queues that do not have many queued (or running) jobs. Check the status of the job to see if it has started or is queued. Tip: use the linux `watch` command to regularly check the status:

```
watch -n 1 <command to watch>
```

Exercise 3; Using multiple nodes

Todo: create example script capable of running on multiple nodes / cores that has a runtime of approx. 2 min

Jobs can use multiple nodes of the cluster simultaneously, as well as multiple cores. Providing an extra argument to `qsub` runs the script (use `example02.sh` for this) on multiple nodes on the cluster[‖]:

```
qsub -t 5 example02.sh
```

Try a `qstat` using the job ID returned by the `qsub` command. As you can see there are multiple jobs running with the same ID. Now take a look at the example script (use `cat` on the `example02.sh` file). There is a dynamic value in this script that results in all jobs executing different command(s). If you want to perform the same actions on different data, this is easier done using the `-t` flag as compared to creating a separate script for each job.

The environment variable `PBS_ARRAYID` contains the value within `[]` shown after the job ID (`qstat` output) and is used to select the data[**]. To get a specific range of values, you can use `-t 25-35` to give a range of values or use `-t 10,15,20,25` to give specific values.

Exercise 4; Modules

Many clusters use a so called *module environment* that is used to make software available to the nodes running the jobs. This requires the user to add the software to their environment before starting a job, or adding a load or add command to the PBS scripts. The module system is very useful for selecting different versions of the same software package. Use the following commands on the cluster that you're logged into now (note: production clusters often contain many more modules):

```
module avail # show available modules
module list  # show loaded modules
```

To add a module to your current environment (unloaded when logging out), use `module add <module_name>` or use `module initadd <module_name>` to load it when you login (requires restarting the session to actually load the module).

Submit the `example04.sh` job to the cluster and wait until it completes. Check the error log file created and use the `module` command to fix the problem and re-submit it to the cluster.

Exercise 5; Dependencies

Another powerful feature of PBS is support for job dependencies. This feature really helps when for instance executing a pipeline that consists of many steps. Instead of placing the commands for all steps into a single PBS script (hard to

[‖]Documentation for *multiple job submission*: `http://www.clusterresources.com/torquedocs/2.1jobsubmission.shtml#jobarrays`

[**]The full list of available environment variables to a job can be seen at: `http://www.clusterresources.com/torquedocs/2.1jobsubmission.shtml#submitvars`

manage, easy to exceed maximum allowed walltime), using dependencies they can stay as separate jobs, submitted simultaneously and wait for the previous job to finish before starting (waiting jobs have the *h* (hold) status when viewing with `qstat`).

Take a look at `example05.sh` which is actually a wrapper script for the jobs `example05a.sh`, `example05b.sh` and `example05c.sh`. As you can see, each job is put into a variable that is referred to in the next step using the:

```
qsub -W depend=afterok:$job_5n ...
```

command. This denotes that the job will start after the previous job has *successfully* completed. Starting multiple jobs when one is finished is possible by reusing the same variable. Other options, such as starting together with a certain job, starting if a job failed (for instance to cleanup after an error), etc, are also possible. See the man page of `qsub` for a list of all types of dependencies.

Instead of submitting the `example05.sh` script to the cluster using `qsub`, you should execute the shell script manually using `sh example05.sh`. You will see from the output that all scripts are submitted to the cluster. Now check the status of the jobs using `showq` or `qstat` and confirm that the remaining jobs are on hold (press Ctrl+C to stop viewing `showq`):

```
watch -n 5 showq -u <user_name>
```

Exercise 6:

Exercise 7:

Exercise 8:

Exercise 9:

Exercise 10:

Discussion

Useful links