Difussion processes on complex networks

Simulation tools - Torque/PBS

Janusz Szwabiński

Intruduction

The goal of this document is to describe the installation and configuration of Torque on a computer that will act as a single-node cluster (which makes sense only in case of multicore CPUs).

The installation was carried out on a Ubuntu 16.04 box. However, the procedure should work on every Debian-like flavor of Linux. Users of other Linux distributions are advised to look for torque packages dedicated to their systems or to install torque from source codes.

Since Torque supports only Linux/Unix platforms, Windows users may want to try one of the following solutions:

- Microsoft HPC Pack (https://msdn.microsoft.com/en-us/library/cc853440(v=vs.85).aspx (https://msdn.microsoft.com/en-us/library/cc853440(v=vs.85).aspx))
- PBS Pro (http://www.pbspro.org/ (http://www.pbspro.org/))

However, due to the famous user-friendliness of all version of Windows these packages will not be covered in this document ;).

Installation

All the following steps have to be done as root (i.e. the superuser) on our computer.

The newest version of Torque (6.1.X) may be downloaded from http://www.adaptivecomputing.com/support/download-center/torque-download/ (http://www.adaptivecomputing.com/support/download-center/torque-download/), and then installed from source. The other possibility is to install the prepackaged binaries. Although the latter usually means an older version of Torque (2.4.16 in case of Ubuntu 16.04), it is sufficient for our purposes and the installation is much easier.

We install all required packages from the command line simply by doing

```
apt-get install torque-server torque-client torque-mom torque-pam
```

Configuration

Cluster configuration

Installing torque sets it up with a default configuration, which is in no way useful. In order to change it we first have to stop all torque services:

```
/etc/init.d/torque-mom stop
/etc/init.d/torque-scheduler stop
/etc/init.d/torque-server stop
```

Next, we kill all instances of `pbs_server` running on our machine (the were started automatically after the installation):

```
killall pbs_server
```

and recreate a clean setup:

```
pbs_server -t create
```

It may happen that at this point you get an error message like:

```
PBS_Server: LOG_ERROR::No such file or directory (2) in chk_file_sec, Sec
urity violation with
"/var/spool/torque/pbs_environment" - /var/spool/torque/pbs_environment c
annot be lstat'd - errno=2,
No such file or directory
PBS_Server: LOG_ERROR::PBS_Server, pbsd_init failed
```

It indicates that something went wrong during the installation and some files (`pbs_environment` in this case) were not properly created. We fix it manually

```
touch /var/spool/torque/pbs_environment
```

and repeat the command

```
pbs_server -t create
```

We will need to answer `yes` here to overwrite the existing database:

```
PBS_Server voyager: Create mode and server database exists,
do you wish to continue y/(n)?y
```

Now we can set up the server process. If your computer has a fully-qualified domain name, use it as the server name. It will be later much easier to add other compute nodes to the cluster. Otherwise you may simply use `localhost` as the server name:

```
echo localhost > /etc/torque/server_name
echo localhost > /var/spool/torque/server_priv/acl_svr/acl_hosts
echo root@localhost > /var/spool/torque/server_priv/acl_svr/operators
echo root@localhost > /var/spool/torque/server_priv/acl_svr/managers
```

We need to tell the server process that the computer itself is a compute node:

```
echo "localhost np=4" > /var/spool/torque/server_priv/nodes
```

And finally, we have to inform the computer node handler (MOM) process which server to contact for work:

```
echo localhost > /var/spool/torque/mom_priv/config
```

Once this is done, we can start the torque processes again:

```
/etc/init.d/torque-server start
/etc/init.d/torque-scheduler start
/etc/init.d/torque-mom start
```

We can check the status of the cluster for instance with the `pbsnodes` command. As result, we should get something like this:

```
localhost
     state = free
     np = 4
     ntype = cluster
     status =  rectime=1496906737,varattr=,jobs=,state=free,netload=367205
687,
     gres=localhost:,loadave=0.44,ncpus=4,physmem=7609320kb,availmem=13510
984kb,totmem=15421412kb,
     idletime=402,nusers=5,nsessions=22,sessions=717 927 978 1488 1497 157
2 1582 1594 1632 1642
     1644 1647 1649 1701 1704 1705 1747 1864 1979 7858 22966 22996,uname=L
inux voyager 4.4.0-78-generic
     #99-Ubuntu SMP Thu Apr 27 15:29:09 UTC 2017 x86_64,opsys=linux
```

Scheduler configuration

The next step is to start the job scheduler:

```
qmgr -c 'set server scheduling = true'
qmgr -c 'set server keep_completed = 300'
qmgr -c 'set server mom_job_sync = true'
```

If you get an `Unauthorized Request` error at this point, you probably messed something up with the server name configuration. Repeat the previous steps carefully. Be sure to use a server name registered in your `/etc/hosts` file.

Queue configuration

We need to create a default queue (it will be called `batch` in the example below):

```
qmgr -c 'create queue batch'
qmgr -c 'set queue batch queue_type = execution'
qmgr -c 'set queue batch started = true'
qmgr -c 'set queue batch enabled = true'
qmgr -c 'set queue batch resources_default.walltime = 1:00:00'
qmgr -c 'set queue batch resources_default.nodes = 1'
qmgr -c 'set server default_queue = batch'
```

You check the status of the queue simply by `qstat -Q` command. Its output should be similar to

```
Queue              Max   Tot   Ena   Str   Que   Run   Hld   Wat   Trn
 Ext T
----------------   ---   ---   ---   ---   ---   ---   ---   ---   ---
 --- -
batch                0     0   yes   yes     0     0     0     0     0
  0 E
```

Server configuration

Finally we need to configure the server to allow submissions from itself:

```
qmgr -c 'set server submit_hosts = localhost'
qmgr -c 'set server allow_node_submit = true'
```

Usage

We submit a single job with the `qsub` command:

```
qsub example.pbs
```

Here, `example.pbs` is a shell script defining required resurces for the task and then executing it. In simulations like those from 6th assigment you often have to sweep a whole parameter space. In this case it is useful to automatically generate a `pbs` script for each parameter set. You may use a simple shell script for that purpose:

In [ ]:

```
#!/bin/bash

wd=/path/to/your/working/directory

for N in 100 200 300; do
  for p in 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0; do
      ( echo "#!/bin/bash"
        echo "#no of cpus"
        echo "#PBS -l nodes=1:ppn=1"
        echo ""
        echo "#name of queue"
        echo "#PBS -q batch"
        echo ""
        echo "#change working directory"
        echo "cd $wd"
        echo "#run the program"
        echo "python3 your_simulation.py -p $p -f 0.5 -n $N" ) > job.sh
        #submit the job
        qsub job.sh
        rm job.sh
  done
done
```

Once you run the script and check the status of the queue again with `qstat`, you get an output like

```
Job id                    Name             User            Time Use S Que
ue
----------------------- ---------------- --------------- -------- - -----
0.localhost            STDIN            szwabin         00:00:00 C batch

1.localhost            job.sh           szwabin                0 R batch

2.localhost            job.sh           szwabin                0 R batch

3.localhost            job.sh           szwabin                0 R batch

4.localhost            job.sh           szwabin                0 R batch

5.localhost            job.sh           szwabin                0 Q batch

6.localhost            job.sh           szwabin                0 Q batch

7.localhost            job.sh           szwabin                0 Q batch

8.localhost            job.sh           szwabin                0 Q batch

9.localhost            job.sh           szwabin                0 Q batch

10.localhost           job.sh           szwabin                0 Q batch
```

If you would start all of the above jobs on single CPU with 4 cores and without Torque installed, the CPU would waste a lot of time for switching between tasks, which would slow down the simulations. Using Torque help to avoid this problem. As you see, there are only 4 processes running, the rest is waiting in the queue. Once one of the running processes is completed, the next one from the queue will be started by the scheduler. Thus, by using Torque you can submit all your simulation jobs in one go and the simply wait for the simulation results. Due to the character of the simulations the speedup on multicore machines should be close to the upper bound given by the Amdahl's law.

One task can be deleted from the queue by

```
qdel job_id
```

If you want to remove all the tasks, use

```
qdelmine
```