Guía de uso

Administrador de colas de trabajo

Daniel Benavides S.

Indice

PBS es un gestor de cola que permite organizar de mejor manera el uso de los recursos con los que cuenta un determinado cluster.

Cada nodo expone una serie de recursos que deben ser administrados y que están disponibles para el uso de los usuarios destinados a ello.

Existiendo múltiples tipos de usuarios, múltiples tipos de recursos, etc, es que se definieron colas de trabajo, las que cuentan y administran distintos recursos, así como permiten el envió de trabajo a uno, varios o todos los usuarios, que requieren recursos de similares características. PBS por defecto genera la cola workq que contiene todos los recursos y que ademas permite que todos los usuarios puedan enviar sus trabajos.

Las colas permiten definir niveles de prioridad, esta prioridad se le asignara a los trabajos, permitiendo discriminar entre trabajos de mas o menos importancia a través de las distintas colas. (Estos valores son solo referenciales y del arbitrio del sistema)

La forma en que los usuarios hacen uso de los recursos es mediante la generación de un script en donde se definen todas las acciones a realizar. Debe considerarse el script como si el usuario estuviera enviando instrucciones mediante una consola del tipo ssh, telnet, u alguna similar.

Existe una serie de valores por defecto, los que se asumen al momento de lanzar los script, pudiendo ser modificados al momento de encolar el script, así como en el propio script con el #PBS iniciando la linea en que se define un determinado parámetro a ser usado por el gestor de colas.

Los scripts pertenecen a una y solo una cola por ejecución, pudiendo ser movidos de cola (Antes de que inicien su ejecución). Lo anterior no quita que el mismo script pueda ser lanzado muchas veces en la misma o distintas colas.

3

Comandos

Para la correcta operación del gestor de colas, se provee de una serie de comandos a nivel de consola, entre estos se encuentras:

qsub

Es el encargado de enviar los script o trabajos a una cola determinada. Se exige unicamente el nombre del script a encolar (Exceptuando el modo interactivo).

```
[dbenavid@feynman ~]$ qsub --help
qsub: invalid option -- '-'
usage: qsub [-a date_time] [-A account_string] [-c interval]
       [-C directive_prefix] [-e path] [-f ] [-h ] [-I [-X]] [-j oe|eo] [-J X-Y[:Z]]
       [-k o|e|oe] [-l resource_list] [-m mail_options] [-M user_list]
       [-N jobname] [-o path] [-p priority] [-P project] [-q queue] [-r y|n]
       [-S path] [-u user_list] [-W otherattributes=value...]
       [-v variable_list] [-V ] [-z] [script | -- command [arg1 ...]]
       qsub --version
[dbenavid@feynman ~]$
```

qdel

Se encarga de detener y eliminar la ejecución de un determinado trabajo.

```
[dbenavid@feynman ~]$ qdel
usage:
       qdel [-W force|suppress_email=X] [-x] job_identifier...
       qdel --version
[dbenavid@feynman ~]$
```

qstat

Permite obtener información del o los trabajos encolados.

Los trabajos terminados no son visibles desde qstat.

Existe un grupo de atributos que permite obtener distinto nivel de segregación de la información de los trabajos. Entre ellos encontramos:

-q : muestra las colas disponibles y sus principales características.

```
[dbenavid@feynman ~]$ qstat -q
```

```
server: feynman

Queue            Memory CPU Time Walltime Node   Run   Que   Lm  State
---------------- ------ -------- -------- ---- ----- ----- ----  -----
workq              --      --       --     --    100     0   --   E R
workq-gpu          --      --       --     --      0     0   --   E R
                                               ----- -----
                                                 100     0
[dbenavid@feynman ~]$
```

-n : muestra todos los nodos a los cuales se ha mandado un job.

```
[dbenavid@feynman ~]$ qstat -n

feynman:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
2428.feynman    dbenavid workq    prueba.pbs  34065   1   1    --    --  R 00:00
   n001/1
[dbenavid@feynman ~]$
```

-u : [usuario] muestra todos los jobs de [usuario].

```
[dbenavid@feynman ~]$ qstat -u dbenavid

feynman:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
2429.feynman    dbenavid workq    prueba.pbs  34097   1   1    --    --  R 00:00
[dbenavid@feynman ~]$
```

-r : muestra la información de los procesos que estan efectivamente corriendo.

-a : muestra los procesos que están en las colas e información adicional.

-f [id] : muestra información detallada de como se lanzo el proceso, tiempo de ejecución, directorio de trabajo e información varia.

```
[dbenavid@feynman ~]$ qstat -f 2431
Job Id: 2431.feynman
    Job_Name = prueba.pbs
    Job_Owner = dbenavid@feynman
    resources_used.cpupercent = 0
    resources_used.cput = 00:00:00
    resources_used.mem = 2976kb
    resources_used.ncpus = 1
    resources_used.vmem = 36076kb
    resources_used.walltime = 00:00:12
    job_state = R
    queue = workq
    server = feynman
    Checkpoint = u
    ctime = Sat Oct 15 19:41:28 2016
```

```
    Error_Path = feynman:/home/dbenavid/prueba.pbs.e2431
    exec_host = n001/1
    exec_vnode = (n001:ncpus=1)
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Sat Oct 15 19:41:29 2016
    Output_Path = feynman:/home/dbenavid/prueba.pbs.o2431
    Priority = 0
    qtime = Sat Oct 15 19:41:28 2016
    Rerunable = True
    Resource_List.ncpus = 1
    Resource_List.nodect = 1
    Resource_List.place = pack
    Resource_List.select = 1:ncpus=1
    stime = Sat Oct 15 19:41:29 2016
    session_id = 39006
    jobdir = /home/dbenavid
    substate = 42
    Variable_List = PBS_O_HOME=/home/dbenavid,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=dbenavid,
        PBS_O_PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr
        /local/pbs/bin:/home/dbenavid/bin,PBS_O_MAIL=/var/spool/mail/dbenavid,
        PBS_O_SHELL=/bin/bash,PBS_O_WORKDIR=/home/dbenavid,PBS_O_SYSTEM=Linux,
        PBS_O_QUEUE=workq,PBS_O_HOST=feynman
    comment = Job run at Sat Oct 15 at 19:41 on (n001:ncpus=1)
    etime = Sat Oct 15 19:41:28 2016
    run_count = 1
    Submit_arguments = prueba.pbs
    project = _pbs_project_default

[dbenavid@feynman ~]$
```

En los ejemplos anteriores básicamente se ven trabajos que están en estado R (Running), sin embargo podemos encontrar otros estados en los que puede estar un trabajo, siendo estos:

Letra de Estado	Estado
E	Trabajo terminando después de haberse ejecutado.
H	Trabajo detenido.
Q	Trabajo encolado y disponible para ser ejecutado.
R	Trabajo en ejecución.
T	Trabajo en transición, Se esta moviendo o re localizando)
W	Trabajo a la espera de que la hora de ejecución se cumpla
S	Trabajo suspendido.

qalter

Permite modificar parámetros de los jobs, pero SOLO cuando se encuentran en el estado Q (trabajo encolado).

```
[dbenavid@feynman ~]$ qalter
usage: qalter [-a date_time] [-A account_string] [-c interval] [-e path]
       [-h hold_list] [-j y|n] [-k keep] [-J X-Y[:Z]] [-l resource_list]
       [-m mail_options] [-M user_list] [-N jobname] [-o path] [-p priority]
       [-r y|n] [-S path] [-u user_list] [-W dependency_list] [-P project_name]
job_identifier...
       qalter --version
[dbenavid@feynman ~]$
```

un ejemplo de ello seria el que se muestra a continuación

```
[dbenavid@feynman ~]$ qstat -u dbenavid

feynman:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
2432.feynman    dbenavid workq    prueba.pbs     --    1   1    --    --  W   --
[dbenavid@feynman ~]$ qalter -l nodes=1:ppn=10 2432
[dbenavid@feynman ~]$ qstat -u dbenavid

feynman:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
2432.feynman    dbenavid workq    prueba.pbs     --    1  10    --    --  W   --
[dbenavid@feynman ~]$
```

Como se ve en el ejemplo, se reemplazo la cantidad de procesadores por nodo de 1 a 10.

pbsnodes

Examina los nodos que se encuentran en el clúster, sus características y cuál es su estado:

```
[nodo]
     state = [estado]
     pcpus = [ncpus]
     jobs = [lista de jobs que tiene ejecutándose]
     …
```

Ejemplo de uso:

```
[dbenavid@feynman ~]$ pbsnodes n001
n001
     Mom = n001
     ntype = PBS
     state = free
     pcpus = 64
     jobs = 2388.feynman/0, 2389.feynman/1, 2390.feynman/2, 2391.feynman/3,
2392.feynman/4, 2393.feynman/5, 2394.feynman/6, 2395.feynman/7, 2296.feynman/60,
2297.feynman/61, 2298.feynman/62, 2299.feynman/63
     resources_available.arch = linux
```

```
     resources_available.host = n001
     resources_available.mem = 65967880kb
     resources_available.ncpus = 64
     resources_available.vnode = n001
     resources_assigned.accelerator_memory = 0kb
     resources_assigned.mem = 12582912kb
     resources_assigned.naccelerators = 0
     resources_assigned.ncpus = 12
     resources_assigned.netwins = 0
     resources_assigned.ngpus = 0
     resources_assigned.vmem = 0kb
     queue = workq
     resv_enable = True
     sharing = default_shared
     license = l


[root@feynman ~]#
```

El nodo `n001` (con 64 cpus y 65 GB de memoria) está disponible para su uso, así también se están ejecutando los trabajos 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2296, 2297, 2298, 2299 observándose que cada uno de esos procesos ocupan solo un procesador (2395.feynman/7), considerando que se están ocupando solo 12 procesadores el state se mantiene en free a la espera de nuevos trabajos.

8

Jobs

Para enviar los trabajos al gestor de colas y este sea capaz de poder saber que acciones son las que tiene que realizar, es que se genera un archivo en donde se especifiquen tanto los recursos que se requieren así como las acciones propias que el usuario desea ejecutar.

Una plantilla de dicho archivo se presenta a continuación:

```
#!/bin/bash
#PBS -N [jobname]
#PBS -l walltime=[HH]:[MM]:[SS]
#PBS -l mem=[MM][kb/mb/gb/tb]
#PBS -q [queueNAME]
#PBS -m [flags (abe)]
#PBS -M [emailCOUNT]
#PBS -e [rutaArchivo]
#PBS -o [rutaArchivo]
#PBS -W depend=afterok:[jobid_1, jobid_2]
#PBS -v [variable]
#PBS -l nodes=[N]:ppn=[M]

cd ${PBS_O_WORKDIR}
…
[aplicación y distintos pasos definidos por el usuario]
…
```

En esta plantilla se muestra la sintaxis de los siguientes directivas PBS:

- **-N [jobname]** : nombre del job
- **-l walltime=[HH]:[MM]:[SS]** : duración del job (en `[horas]:[minutos]:[segundos]`). (Opcional: asumirá por defecto el especificado en el sistema)
- **-l mem=[MM][kb/mb/gb/tb]** : memoria requerida y límite de la memoria (número entero) en kb: kilobytes, mb: megas, gb: gigas, tb: teras. (Opcional: asumirá por defecto el especificado en el sistema)
- **-q [queue]** : nombre de la cola a la cual se manda el job. (Opcional: asumirá por defecto el especificado en el sistema)
- **-m [flags]** : indica cuando se tiene que mandar un correo. Si no se pone este requerimiento si el job es abortado por el sistema se manda un correo al usuario (variable `MAIL`). Hay las siguientes opciones (son combinables):
    - **n**: sin correo
    - **a**: el job es abortado por el sistema
    - **b**: se inicia la ejecución del job
    - **e**: el job a terminado
- **-M [emailCOUNT]**: dirección de correo donde se quiere que mande un job
- **-e [rutaArchivo]**: ruta del archivo en donde se quiere almacenar la salida estandard del error del job (si no se indica se construye en el directorio donde está el script el archivo ${PBS_JOBNAME}.e${PBS_JOBID})

- **-o [rutaArchivo]**: ruta del archivo en donde se quiere almacenar la salida estandard del job (si no se indica se construye en el directorio donde está el script el archivo ${PBS_JOBNAME}.o${PBS_JOBID})
- **-W depend=afterok:[jobid]**: el job se mandará a la cola cuando otro job anterior (el número [jobid]) termine su ejecución
- **-v [variable]**: para mandar un job que tome el valor `[variable]`
- **-l nodes=[N]:ppn:[M]**: número de nodos (`[N]`) y número de cpus a coger de cada nodo (`[M]`) la aplicación se repartirá en [N]x[M] cpus
- **-a [hora]:** `[[[[CC]YY]MM]DD]hhmm[.SS] donde CC=centuria, YY=Año, MM=mes, DD=día, hh=hora, mm=minuto, SS=segundo`

Y para su envío se utilizaría el comando `qsub` seguido del script:

```
[user@server ~]$ qsub job.pbs
```

O haciendo uso de los los flags del comando:

```
qsub -N [jobname] -l walltime=[HH]:[MM]:[SS] -l mem=[MM] -q [queueNAME] -m [flags] -M
[emailCOUNT] -e [rutaArchivo] -o [rutaArchivo] -W afterok:[jobid] -v [variable] -l
nodes=[N]:ppn=[M] [script]
```


10

Distintos tipos de lanzamiento de jobs

Lanzar un job interactivo

A la hora de hacer pruebas es muy útil abrir una sesión interactiva en un nodo del clúster. Esto se hace mandando un job interactivo. La sesión que se abra durará todo el tiempo que se quiera hasta que no se le mande la instrucción de salida `'exit'`. La sintaxis es (la cola asume '`ppn=1`'):

```
qsub -I -l nodes=1 -q [queue]
```

A la hora de lanzar este tipo de jobs se tiene que ser muy consciente de que se está ocupando un nodo del clúster.

Lanzar un job con dependencias

En este caso, no se lanzará un script listar.pbs hasta que no termine la espera de 60 segundos:

```
[dbenavid@feynman ~]$ qsub prueba.pbs
1341.feynman
[dbenavid@feynman ~]$ qsub -W depend=afterok:1341 listar.pbs
1342.feynman
[dbenavid@feynman ~]$ qstat -a

feynman:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1338.feynman    alara    workq    GMX_npt_ch  17686   1  20 1024mb   --  R 00:16
1341.feynman    dbenavid workq    prueba.pbs  17819   1   1    --    --  R 00:00
1342.feynman    dbenavid workq    listar.pbs    --    1   1    --    --  H   --
[dbenavid@feynman ~]$
```

El script `'listar.pbs'` no se ejecutará hasta que termine `'prueba.pbs'` con identificador 307079.

Trabajo con variable

Tomamos el job sencillo de ejemplo. En este caso el tiempo de espera para la script se lo pasaremos como variable del job.

El script.sh es:

```
#!/bin/bash
### Job name
#PBS -N prueba
### Max run time
#PBS -l walltime=00:00:10
### Queue name
#PBS -q workq
### Number of nodes and processors per node
#PBS -l nodes=1:ppn=1

cd ${PBS_O_WORKDIR}
date
sleep ${espera}
date
```

El script ejecuta de la siguiente manera:

```
[dbenavid@feynman ~]$ qsub -v espera=120 script.sh
2398.feynman
[dbenavid@feynman ~]$ qstat -u dbenavid

feynman:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
2398.feynman    dbenavid workq    prueba      12513   1   1    --  00:00 R 00:00
[dbenavid@feynman ~]$
```


12

Ejemplos de script

ej1.sh

```
date
```

Esta es la expresión mas básica que se puede encontrar como un script, ya que unicamente ejecuta un comando simple, asumiendo todos los valores por defecto.

ej2.sh

```
#!/bin/bash
#PBS -N ej2
date
sleep 10
date
```

Podemos encontrar que se se setean variables del trabajo, como por Ej el nombre con el que se podra ver en la lista de trabajos

ej3.sh

```
#!/bin/bash
#PBS -N ej3
pwd
ls -lh
```

En este ejemplo se pretende ilustrar la ruta de trabajo por defecto que asume el trabajo al momento de iniciar su ejecución.

ej4.sh

```
#!/bin/bash
#PBS -N ej4
set | grep PBS
```

El ejemplo entrega por el archivo que recibe la salida estándar, las variables que genera el PBS cuando inicia la ejecución de la tarea.

ej5.sh

13

```
#!/bin/bash
#PBS -N ej5
pwd
cd $PBS_O_WORKDIR
pwd
```

Se muestra como poder cambiar el directorio de trabajo a la misma ubicación desde la que se lanzo el trabajo, siendo esta opción útil ya que facilita la generación de los script al poder ir ejecutando pequeñas acciones las que se le van agregando al script sin tener que alterar ruta alguna.

ej6.sh

```
#PBS -N ej6
#PBS -q workq
#PBS -l nodes=2:ppn=3
NPROC=`wc -l < $PBS_NODEFILE`
source /opt/shared/openmpi-2.0.1/environment.sh
cd $PBS_O_WORKDIR
mpiexec -np $NPROC prueba_mpi
```

Cuando se trabaja con múltiples nodos o mas de un procesador, se hace necesario conocer el número total de procesadores a usar, en este caso se obtiene dicho número, almacenándose en la variable ***NPROC*** la que se obtiene del archivo que se contiene los host (***$PBS_NODEFILE***) y la cantidad de cores que se ha de usar

ej7.sh

```
#!/bin/sh

#PBS -N ej7
#PBS -l nodes=1:ppn=64
#PBS -l mem=1GB
#PBS -q workq

ID=`echo $PBS_JOBID | cut -d. -f1`
NPROC=`wc -l < $PBS_NODEFILE`
cd $PBS_O_WORKDIR
unset OMP_NUM_THREADS
. /opt/shared/openmpi-2.0.1/environment.sh
export BASENAME=topol
export ORCA_PATH=/opt/shared/orca_3_0_2_linux_x86-64
export GMX_QM_ORCA_BASENAME=topol
export GMX_ORCA_PATH=/opt/shared/orca_3_0_2_linux_x86-64
export SCRDIR=/scratch/${ID}

mkdir $SCRDIR
cp -r * $SCRDIR/
cd $SCRDIR
BINDIR=/opt/shared/gmx-5.1.4/bin
MY_PROG="${BINDIR}/gmx mdrun -nt ${NPROC} -s topol.tpr "
$MY_PROG > $PBS_O_WORKDIR/daniel.out 2>&1 > $PBS_O_WORKDIR/output.log
cp -r * $PBS_O_WORKDIR/
rm -rf $SCRDIR
```


14

En ocasiones cuando se corren procesos en un solo nodo (Distinto al nodo líder), es mas eficiente mover los datos con los que se han de trabajar, cosa de que todo el proceso de *Lectura/Escritura* a disco se efectúe localmente, evitando de esta manera la saturación de la red así como el cuello de botella que se produciría al escribir todos los trabajos sobre el(los) mismo(s) disco(s). Bajo este concepto es que el script, genera un directorio en una carpeta local cuyo nombre depende del id del trabajo. Luego se copian los datos desde el directorio de trabajo, al directorio temporal. Se ejecuta el proceso requerido. Se devuelven los datos al directorio de trabajo una vez que se termine la ejecución del proceso. Se libera el espacio ocupado con la carpeta temporal.

Con el fin de darle seguimiento al avance del proceso, se redirige la salida estándar a un archivo alojado en la carpeta en el home desde donde se lanzo el trabajo (***PBS_O_WORKDIR***).

15

Administración del servidor

Para la administración de las colas, los nodos, usuarios y otra gran cantidad de opciones, es que existe el comando ***qmgr***.

Las acciones mas usadas que se pueden realizar con el qmgr son:

Acción	Descripción
active	Activa el objeto en cuestión (queue, node, server, resource)
create	Crea el objeto en cuestión (queue, node, resource)
delete	Elimina el objeto en cuestión (queue, node, resource)
set	Permite especificar valores a determinados atributos del objeto (queue, node, server, resource)
unset	Quita de definición de un determinado atributo del objeto (queue, node, server, resource)
list	Lista los atributos de un objeto (queue, node, server, resource)
print	Muestra los comandos necesarios para regenerar el objeto a su condición actual (queue, node, server, resource)
help	Muestra como se usan los comandos
quit	Permite salir del qmgr

Existen dos formas de invocar qmgr, una es simplemente invocándolo y entrando a su propio shell, en donde se pueden ir ejecutando una a una las acciones requeridas y la segunda manera es mediante la shell del sistema operativo e ir ejecutando cada una de las acciones requeridas. A continuación se expondrá mas extensamente dichas opciones

16

Ejecución de qmgr mediante su propia shell

Con el fin de simplificar el uso qmgr proporciona su propia shell, logrando con ello generar un ambiente de uso exclusivo evitando posibles confusiones con lo que es propio del sistema operativo.

```
[dbenavid@feynman ~]$ qmgr
Max open servers: 49
Qmgr: print queue workq
#
# Create queues and set their attributes.
#
#
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
Qmgr: quit
[dbenavid@feynman ~]$
```

En el ejemplo anterior se muestran los pasos que se deben ejecutar para dejar la cola workq desde 0 a su condición actual. Todo ello dentro de la shell de qmgr.

Ejecución mediante la shell del sistema operativo

En ocasiones se requiere automatizar ciertas acciones, ya sea en un crontab o por casos varios; ante ello qmgr proporciona un método interactivo de ejecutar los comandos, básicamente se pasa el comando a ejecutar en qmgr a tráves del parámetro **-c**.

```
[dbenavid@feynman ~]$ qmgr -c "print queue workq"
#
# Create queues and set their attributes.
#
#
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
[dbenavid@feynman ~]$
```

Como se aprecia en el ejemplo anterior vemos que el comando a ejecutar en qmgr es pasado por parámetro, sin embargo el resultado entregado es exactamente igual al entregado en el ejemplo de la sección “***Ejecución de qmgr mediante su propia shell***”.

17

Manejo de colas (queue)

Las colas son el ente por el que se gestiona la entrega de recursos a un determinado trabajo.

PBS provee dos tipos de cola; las de tipo ***Execution*** y las de tipo ***Route***. Las colas de tipo Execution, como su nombre lo indica, son las que reciben los trabajos listos para ser ejecutados y definen las políticas de privilegios, accesos etc. Las de tipo Route permite enviar a trabajos a una cola predeterminada, y dependiendo de los recursos solicitados se deriva dicho trabajo a una cola definida.

En los ejemplo que se viene se presenta un caso de como se entrelazan colas de Execution y de Route.

```
create queue submit
set queue submit queue_type = Route
set queue submit route_destinations = server_1
set queue submit route_destinations += server_2
set queue submit route_destinations += cluster
set queue submit enabled = True
set queue submit started = True

create queue server_1
set queue server_1 queue_type = Execution
set queue server_1 from_route_only = True
set queue server_1 resources_max.arch = linux
set queue server_1 resources_min.arch = linux
set queue server_1 acl_group_enable = True
set queue server_1 acl_groups = math
set queue server_1 acl_groups += chemistry
set queue server_1 acl_groups += physics
set queue server_1 enabled = True
set queue server_1 started = True

create queue server_2
set queue server_2 queue_type = Execution
set queue server_2 from_route_only = True
set queue server_2 resources_max.arch = sv2
set queue server_2 resources_min.arch = sv2
set queue server_2 acl_group_enable = True
set queue server_2 acl_groups = math
set queue server_2 acl_groups += chemistry
set queue server_2 acl_groups += physics
set queue server_2 enabled = True
set queue server_2 started = True

create queue cluster
set queue cluster queue_type = Route
set queue cluster from_route_only = True
set queue cluster resources_max.arch = linux
set queue cluster resources_min.arch = linux
set queue cluster route_destinations = long
set queue cluster route_destinations += short
set queue cluster route_destinations += medium
set queue cluster enabled = True
set queue cluster started = True

create queue long
set queue long queue_type = Execution
set queue long from_route_only = True
set queue long resources_max.cput = 20:00:00
set queue long resources_max.walltime = 20:00:00
set queue long resources_min.cput = 02:00:00
```


18

```
set queue long resources_min.walltime = 03:00:00
set queue long acl_group_enable = True
set queue long acl_groups = astrology
set queue long enabled = True
set queue long started = True

create queue short
set queue short queue_type = Execution
set queue short from_route_only = True
set queue short resources_max.cput = 01:00:00
set queue short resources_max.walltime = 01:00:00
set queue short enabled = True
set queue short started = True

create queue medium
set queue medium queue_type = Execution
set queue medium from_route_only = True
set queue medium enabled = True
set queue medium started = True
set server default_queue = submit
```

Una lista mas completa de los recursos para limitar se puede encontrar en la sección 5.4 del manual "***Reference Guide***" que se puede obtener de:
http://www.pbsworks.com/pdfs/PBSReferenceGuide13.0.pdf

19