infocluster Roll: Users Guide

1.0 Edition

infocluster Roll: Users Guide :
1.0 Edition
Published Feb 03 2022

Table of Contents

List of Tables

Prefacio

Este documento presenta como usar el cluster.

 Este documento esta escrito sin acento!

Por favor visite la pagina Teopanzolco[1] para conocer lo ultimo sobre el estado del cluster.

Notes

1. http://teopanzolco.ibt.unam.mx

Chapter 1. Presentacion del cluster

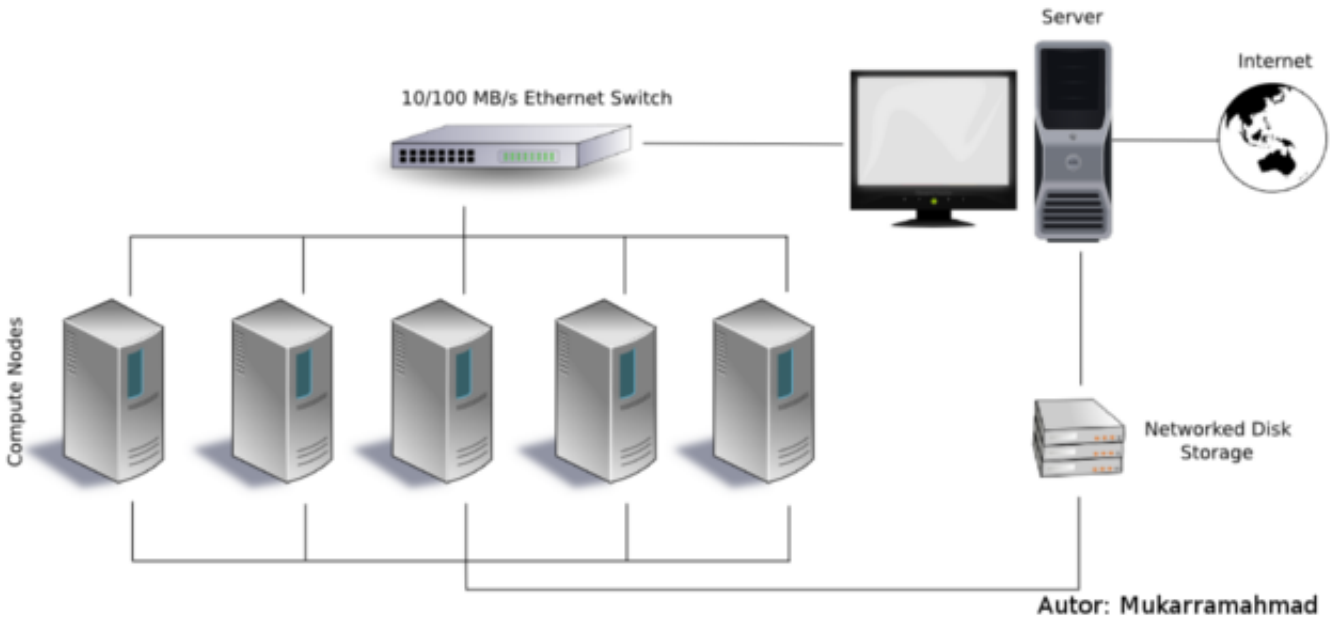
1.1. Esquema de un cluster

El cluster del IBt es basado en un cluster Beowulf[1]. Se basa en :

- Una maquina **maestra** : *master*
- Maquinas para realizar las tareas de calculo :*nodos de calculo*
- Una red de administraciÃ³n *Ethernet*.
- Servidor(es) de almacenamiento de datos (*opcional*).
- Una red de datos, como InfiniBand (*opcional*).

Un cluster *Beowulf* es una supercomputadora de memoria distribuida, tipo MIMD (**M**ultiple **I**nstruction **M**ultiple **D**ata)

Presentamos un esquema de un cluster tipo Beowulf:



En lo general, la maquina *master* es la que utilizan los usuarios del cluster para administrar sur datos y sus trabajos. No se permite utilizar esta maquina para realizar tareas computacionalemente intensas, como analysis de datos.

1.2. Equipo

El nuevo cluster del IBt *Teopanzolco* cuenta con:

- El master (Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz) de 48 cores y 64 GB de memoria
- 5 nodos (AMD Opteron(tm) Processor 6376) de 64 cores y 512 GB de memoria
- 1 nodo (AMD Opteron(tm) Processor 6376) de 64 cores y 1 TB de memoria
- Red Infiniband (IB) a 40Gb/s
- Sistema de archivos *Lustre* para el **scratch** (55 Tb en total)
- Manejador de cola **Sun Grid Engine** (SGE)

Aqui se muestra el rack principal del cluster:

Notes

1. https://en.wikipedia.org/wiki/Beowulf_cluster

Chapter 2. Utilizar el cluster

2.1. Conectarse al cluster

Como cualquier maquina UNIX, se necesita una cuenta de usuario para poder utilizar el cluster. Esta cuenta se pide escribiendo al correo electronico del IBt **clustersupport** at **ibt.unam.mx**. Recibiran todos los detalles de su cuenta en un correo.

Por el momento, se puede acceder al cluster unicamente via el protocolo Secure Shell. Suponiendo que ya tienen su cuenta **usuario** con la contrasena **XXXXXX**. Ademas, si son usuarios de Linux o MacOSX, simplemente abren un terminal y teclean lo siguiente:

```
$ ssh -l usuario teopanzolco.ibt.unam.mx
Password:    XXXXXX          ( <---- teclean su contrasena)
Rocks 6.2 (SideWinder)
Profile built 16:36 02-Oct-2015

Kickstarted 11:55 02-Oct-2015

El cluster con el que usted estÃ¡ trabajando existe gracias al esfuerzo del Instituto de Biotecno
a.
Les pedimos que cualquier publicaciÃ³n o trabajo realizado en parte gracias al uso del mismo, sea
era siguiente (en ingles):

"The computer analysis was performed using the cluster of the Instituto de Biotecnologia-UNAM."
o
"We thank the Instituto de Biotecnologia-UNAM for giving us access to its computer cluster."

Cualquier duda o aclaracion: escribir a "clustersupport@ibt.unam.mx"
Buen trabajo!
[usuario@teopanzolco ~]$
```

En caso de ser la primera vez que se conecten desde su maquina propia, puede aparecer un mensage de advertencia similar a este:

```
$ ssh -l usuario teopanzolco.ibt.unam.mx
The authenticity of host 'teopanzolco.ibt.unam.mx (132.248.32.104)' can't be established.
RSA key fingerprint is e9:2c:10:4a:54:24:fc:43:23:eb:89:99:b0:42:98:cf.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'teopanzolco.ibt.unam.mx,132.248.32.104'
(RSA) to the list of known hosts.

  ../..
```

Para poder seguir con el proceso de coneccion, deben de contestar **yes**. Esta alerta surge porque no esta registrado de manera oficial (tal como un sitio de un banco o de venta en linea), el cluster. Asi que confien en que el nombre de la maquina que se indica en el comando ssh sea verdadera.

Los usuarios de Windows pueden usar la paqueteria PuTTY[1]. No se presentara aqui como usar este programa, pero pueden pedirnos ayuda en caso de dudas.

2.2. Scheduler SGE

2.2.1. Rol du scheduler

El scheduler se encarga, dentro de un cluster, de organizar los trabajos de los usuarios. El scheduler del cluster es Sun Grid Engine (SGE).

- Mantiene la informaciÃ³n de los recursos y sus disponibilidades para actualizar las colas.
- Recibe las peticiones de los usuarios (job) y las ordena por prioridad.
- Determina cual job puede correr en donde cuando los recursos estÃ¡n disponibles.
- Se encarga de revisar los estados de estos trabajos.

2.2.2. Colas del cluster

El cluster cuenta con 3 colas pricinpales.

- Cola **all.q** : Trabajos de menos de 24 horas reales.
- Cola **express.q** : Trabajos interactivos . No mas de 2 horas reales.
- Cola **lenta.q** : Trabajos de mas de 24 horas reales.

La cola **all.q** tiene mas prioridad que la **lenta.q**. Ademas, cuenta generalmente con mas cores disponibles. Revise el parafo *Opciones usuales* para ver como pedir cores en la cola **lenta.q**.

La cola **express.q** es dedicada para pruebas de corta duraciÃ³n. En ella pueden correr unicamente jobs interactivos, como se muestra en el parafo Job interractivo.

2.2.3. El buen uso del cluster

Como vimos, el master del cluster es para administrar los datos, enviar los jobs, revisar sus estados y los resultados.

Son los nodos que realmente realizan las tareas fuertes. Asi que:

- No se debe de usar el mÃ¡ster del cluster para trabajos pesados.
- No se debe conectarse directamente a un nodo para realizar sus tareas: usan un job interactivo.
- No esperen a que unos recursos sean disponible para mandar sus jobs!
- Se hace un chequeo cotidiano y se arreglan las colas segÃºn los jobs en espera.

Los usuarios que no respetan esta regla veran sus trabajos indebidos suprimidos al inicio, y se podra bloquear la cuenta en caso de seguir mal usando los recursos.

2.2.4. Estado de las colas

El comando para ver el estado de las colas es **qstat -g c** :

```
$ qstat -g c
CLUSTER QUEUE                   CQLOAD   USED    RES  AVAIL  TOTAL aoACDS  cdsuE
--------------------------------------------------------------------------------
all.q                             0.03    0    0   0    320 0       0
lenta.q                           0.03    0    0   0    20 0       20
```

En este caso hay 320 cores disponibles para la cola normal, y 0 (de 20 en total) para la cola lenta. Y el cluster no esta usado para nada (0 jobs en total corriendo).

```
$ qstat -g c
CLUSTER QUEUE                   CQLOAD   USED    RES  AVAIL  TOTAL aoACDS  cdsuE
--------------------------------------------------------------------------------
all.q                             0.98    320    0   0    320 0       0
lenta.q                           0.98    0    0   0    20 0       20
```

En este caso los 320 cores estan todos utilizados por la cola normal. Notamos que la carga es casi 1, es decir, todos los cores del cluster estan trabajando.

2.2.5. Prioridad de los jobs

Se determino una polÃtica de prioridad (especifica para el cluster del IBt). a prioridad se calcula en cada ciclo (30 segundos), y se determina asÃ:

- El que mas lo usa, menos prioridad tiene.
- La cola normal tiene mas prioridad que la cola lenta.
- Cada job en espera tendrÃ¡ un valor de prioridad entre [0,1]. Un valor de 1 es mas prioritario que un valor de 0.

Revisen el capitulo siguiente Escribir jobs para tener mas detalle sobre el contenido de un job.

2.3. Scratch en el cluster

El home de cada usuario esta limitado en tamaÃ±o. Se autoriza una ocupaciÃ³n de 20GB, llegando a los 25GB por no mÃ¡s de una semana.

Para conocer cuanto estan ocupando y sus limites, pueden usar el comando **quota**. La opciÃ³n **-s** es para tener un formato mÃ¡s leible.

```
$ quota -s
Disk quotas for user jerome (uid 1865):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
```

```
10.2.255.249:/export/home/jerome
                  1072M  20480M  25600M           21772       0       0
```

En este caso, vemos que el limite es de 20480MB, y la ocupaciÃ³n actual del usuario es de 1072Mb.

Para poder trabajar, el cluster cuenta con dos tipos de espacios para almacenar los datos temporales:

- **/state/partition1** en cada nodo de cÃ¡lculo.
- **/scratch** comÃºn a todo el cluster (espacio de tipo Lustre).

Presentamos cada uno para que sepan como utilizarlos.

2.3.1. /state/partition1

Cada nodo tiene una particiÃ³n de scratch, dentro de su disco duro propio. Esta particiÃ³n se llaman /state/partition1 , y es dedicada al almacenamiento de los datos. La regla de uso es que para evitar conflictos, cada usuario crea su propio directorio (con su nombre por ejemplo). Por ejemplo, en seguida viene un extracto de un script que define el temporal para los datos en el nodo. Ntan que usamos ademÃ¡s del nombre del usuario, el numero del job. Eso permite separar aÃºn mas los datos, en caso de estudios similares.

```
Temp="/state/partition1/$USER/${JOB_ID}/"
mkdir -p $Temp
cd  $Temp

.../..

# al final lo borramos
cd
rm -rfv $Temp
```

2.3.2. /scratch

El cluster dispone de un almacenamiento visibles en todos los nodos, usando el sistema de archivos Lustre [2]. Este sitema se basa en la red IB para la transferencia de los datos. Aunque es un poco menos veloz que la transferencia con un disco duro local, la ventaja es que se pueden acceder a los datos desde cualquier maquina del cluster.

Es espacio disponible suma un total de 52TB. Para que no existen conflictos en los nombre de archivos, la buena regla es usar este espacio, creando un subdirectorio (con su nombre de usuario mejor) para alli depositar los archivos. Retomando el ejemplo anterior, podemos usar el scratch de esta manera:

```
Temp="/scratch/$USER/${JOB_ID}/"
mkdir -p $Temp
cd  $Temp

.../..

# al final lo borramos
cd
rm -rfv $Temp
```

Este espacio no es respaldado. Al momento que sea necesario, se limpiaran primero los archivos mÃ¡s viejos, para dejar este espacio con menos de 85% de ocupaciÃ³n total. AsÃ que tomen sus precauciones para evitar cualquier perdida de datos importantes.

Notes

1. http://www.putty.org/
2. http://lustre.org/

Chapter 3. Escribir jobs para el cluster

3.1. Job Interactivo

Un job interactivo permite al usuario acceder a recursos y probar como funciona su programa. AsÃ se debe de acceder a los recursos disponibles al momento de probar de manera interactiva un programa. Porque este tipo de job entra en competencia con los demÃ¡s jobs esperando recursos.

No se debe de usar este tipo de job para tareas complejas y largas. Por defaut, la cola de mas de 24 horas no permite este tipo de jobs.

El mÃ©todo para trabajar con un job interactivo se basa en el uso de los comandos **qrsh** o **qlogin**. La diferencia entre los dos comandos es muy sutil, asÃ que no entra en cuenta para este documento.

Se presenta un ejemplo de uso de qrsh. Para mas claridad sobre como funciona, se dejo tal cual el prompt al inicio de cada linea de comando. Al momento de pedir un nodo con un solo core, el sistema de colas entrega el nodo **compute-0-4**. Y es importante salir, una vez las pruebas realizadas, con el comando **exit**. De lo contrario, se quedara el job interactivo en este nodo por no mas de 24 horas.

```
[jerome@teopanzolco ~]$ qrsh
[jerome@compute-0-4 ~]$ hostname
compute-0-4.local

../..   # ejecuciÃ³n de las pruebas

[jerome@compute-0-4 ~]$ exit
logout
[jerome@teopanzolco ~]$
```

Por su definiciÃ³n misma, un job interactivo trata de acceder a el recurso pedido, por no mas de 30 segundos. Si el cluster esta muy cargado y no hay disponibilidad al momento de un recurso, se puede usar la opciÃ³n **now no** para autorizar el job a quedarse encolados. Pero se debe de usar con cuidado, porque el job podrÃa entrar en cualquier momento, Â¡ hasta cuando el usuario no este al frente de su computadora!

3.2. Job simple

Por lo general, utilizar el scheduler SGE se basa en escribir un script que describe la tarea. Una vez este script correcto, se manda a la cola deseada y se espera a que la tarea se realiza en el cluster. Puede ser una tarea compleja y pesada esta escritura. Por eso es una buena costumbre basarse en scripts anteriores que se sabe que funcionan, y adaptarlos a la nueva tarea.

En el mÃ¡ster del cluster, se proporciona al usuario unos ejemplos de scripts, en el directorio **/opt/gridengine/examples/jobs**. Se muestra aquÃ un ejemplo de un job simple. Los comentario empiezan con el carÃ¡cter **#**. Las opciones para el scheduler empiezan con **#$** seguido de la opciÃ³n.

```
#!/bin/bash
# This is a simple example of a SGE batch script

# Nombre del job
```

```
#$ -N Simple

#
# print date and time
date
# Sleep for 20 seconds
sleep 20
# print date and time again
date
```

Para correr el job, se utiliza el comando **qsub**. Las opciones del comando, como vimos con el ejemplo, pueden integrarse dentro del script, o directamente en linea de comando. Y la ultima palabra la tiene las opciones en linea.

Lo que sigue muestra la manera general de someter un job, y verificar su estado:

```
$ qsub simple.sh
Your job 51 ("Simple") has been submitted
$ qstat
job-ID  prior   name       user         state submit/start at     queue
---------------------------------------------------------------------------------------------
     51 0.00000 Simple     jerome       qw    02/16/2016 12:54:58
$ qstat
job-ID  prior   name       user         state submit/start at     queue
---------------------------------------------------------------------------------------------
     51 0.55500 Simple     jerome       r     02/16/2016 12:55:08 all.q@compute-0-0.local
$ qstat
$
```

El job mandado tiene el numero 51, que sera su **JOBID** para el sistema de colas SGE. Luego, se verifica su estado, y esta encolado y esperando (**qw**). Sin opciÃ³n, el comando **qstat** muestra los jobs para el usuario. Para ver a los jobs de todos los usuarios, se necesita la opciÃ³n **-u ***.

El job entra a correr (**r**). A partir de este momento, debido a las caracterÃsticas de la cola, tiene 24 horas para terminar. Como lo vimos, el script es muy simple, asÃ que en menos de un minuto acabara. Y cuando esto ocurre, el comando qstat no enseÃ±a ningÃºn job.

Al final, se obtiene 2 archivos, que contienen la salida estÃ¡ndar y la salida de error, con nombre **oJOBID** y **eJOBID** respectivamente. En este caso preciso, como se puso un nombre al job (opciÃ³n **-N**), se obtendrÃ¡ 2 archivos. Se verifica que el script funciono (la salida de error esta vacÃa), y que el script hizo lo que se le pidiÃ³:

```
$ ls -l | grep Simple
-rw-r--r--  1 jerome jerome        0 16 fÃ©vr. 12:55 Simple.e51
-rw-r--r--  1 jerome jerome      142 16 fÃ©vr. 12:58 Simple.o51
$ cat Simple.o51
Tue Feb 16 12:55:09 CST 2016
Tue Feb 16 12:55:29 CST 2016
```

3.3. Job con Blast

Sabiendo como funcionan los script, se les propone ahora un script un poco mas Ãºtil. Se basa en una bÃºsqueda con Blast de secuencias de nucleÃ³tidos contenidas en de un archivo fasta **nuc.fasta**. Buscamos en una base de datos **nt** (ya formateada), y ponemos el resultado en un archivo **results.txt**

```
#!/bin/bash

# Nombre del job
#$ -N  Blastn-NT

# Para obtener el PATH adecuado
source /share/apps/Profiles/share-profile.sh

# UbicaciÃ³n de las bases de datos formateadas.
export BLASTDB=/scratch/BlastDB/

OutFile="results.txt"

blastn -db nt -query nuc.fasta -out $OutFile
```

Se manda el job, y se checa su estado. Hasta verificar el archivo resultado:

```
$ qsub blast.sh
Your job 250 ("Blastn-NT") has been submitted
$ qstat -j 250
==============================================================
job_number:                 240
exec_file:                  job_scripts/250
submission_time:            Tue Feb 16 17:05:40 2016
owner:                      jerome
uid:                        1865
group:                      usmb
gid:                        503
sge_o_home:                 /home/jerome
sge_o_log_name:             jerome
sge_o_path:                 /opt/openmpi/bin:/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/
sge_o_shell:                /bin/bash
sge_o_workdir:              /home/jerome
sge_o_host:                 teopanzolco
account:                    sge
mail_list:                  jerome@teopanzolco.local
notify:                     FALSE
job_name:                   Blastn-NT
jobshare:                   0
env_list:
script_file:                Scripts/script-test.sh
  ../..
$ qstat -j 250
Following jobs do not exist:
250
$ head -30 results.txt
BLASTN 2.3.0+
```

```
Reference: Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb
Miller (2000), "A greedy algorithm for aligning DNA sequences", J
Comput Biol 2000; 7(1-2):203-14.



Database: nt
           34,389,867 sequences; 110,383,924,525 total letters



Query= gi|457872547|ref|XM_004223145.1| Plasmodium cynomolgi strain B
hypothetical protein (PCYB_112670) mRNA, partial cds

Length=1190
                                                                      Score     E
Sequences producing significant alignments:                          (Bits)  Value

  gi|457872547|ref|XM_004223145.1| Plasmodium cynomolgi strain B ...  2198    0.0
  gi|672188562|ref|XM_008816634.1| Plasmodium inui San Antonio 1 ...  1173    0.0
  gi|156101254|ref|XM_001616271.1| Plasmodium vivax SaI-1 hypothe...  1123    0.0
  gi|817745457|ref|XM_012482260.1| Plasmodium fragile hypothetica...  1098    0.0
  gi|194247187|emb|AM910993.1| Plasmodium knowlesi strain H chrom...  1098    0.0
  gi|221057699|ref|XM_002261322.1| Plasmodium knowlesi strain H h...  1096    0.0


> gi|457872547|ref|XM_004223145.1| Plasmodium cynomolgi strain
```

Para los curiosos, se puede obtener las estadÃsticas de uso del job. El comando **qacct** indica informaciones como memoria, tiempo CPU, y otros detalles:

```
$ qacct -j 250
==============================================================
qname        all.q
hostname     compute-0-4.local
group        usmb
owner        jerome
project      NONE
department   defaultdepartment
jobname      Blastn-NT
jobnumber    250
taskid       undefined
account      sge
priority     0
qsub_time    Tue Feb 16 17:05:40 2016
start_time   Tue Feb 16 17:05:52 2016
end_time     Tue Feb 16 17:08:32 2016
granted_pe   NONE
slots        1
failed       0
exit_status  0
ru_wallclock 160
ru_utime     50.350
ru_stime     109.798
ru_maxrss    3844740
```

```
ru_ixrss     0
ru_ismrss    0
ru_idrss     0
ru_isrss     0
ru_minflt    6840092
ru_majflt    12
ru_nswap     0
ru_inblock   45616
ru_oublock   64
ru_msgsnd    0
ru_msgrcv    0
ru_nsignals  0
ru_nvcsw     492
ru_nivcsw    16236
cpu          160.149
mem          512.448
io           0.000
iow          0.000
maxvmem      3.836G
arid         undefined
```

3.4. Job Array

3.4.1. Generalidades

Por lo general, se presenta el caso en lo cual tenemos que mandar un numero largo de jobs, que son casi idÃ©nticos en los comandos a correr, cambiando por algÃºn parÃ¡metro. Por ejemplo, un conjunto de 1000 grupos de secuencias que se requiere analizar con Blast. La primera idea suele ser escribir 1000 jobs, y mandarlos a correr. AllÃ es donde el **Job Array** ayuda.

El job Array es un script que sera ejecutado varias veces. Cada ejecuciÃ³n tendrÃ¡ los mismos derechos que un script normal, en cuanto a tiempo, memoria , etc... La diferencia entre cada ejecuciÃ³n sera el valor de una variable de ambiente **SGE_TASK_ID**. Esta a cargo del script del job a utilizar de manera adecuada este valor para realizar la tarea correspondiente. La opciÃ³n de qsub para iniciar un Job Array es **-t** [1]

```
$ qsub -t 1-1000 miscript.sh
Your job-array 53.1-1000:1 ("TestArray") has been submitted
```

El script miscript correrÃ¡ 1000 veces, la primera vez el valor de SGE_TASK_ID sera de 1, luego valdrÃ¡ 2, y asÃ enseguida hasta llegar a 1000. Si se requiere por una razÃ³n especifica, cada tercer numero, se usara la opciÃ³n -t de esta manera. Y se verifica el estado de los jobs :

```
$ qsub -t 1-1000 miscript.sh
Your job-array 54.1-1000:4 ("TestArray") has been submitted
$ qstat
job-ID  prior   name       user         state submit/start at     queue
---------------------------------------------------------------------------------------------
53 0.55500 TestArray  jerome       qw    02/17/2016 11:37:46                                    1
```

```
54 0.55500 TestArray  jerome       qw    02/17/2016 11:38:29                                    1
```

Como se puede notar, un job Array tiene un solo numero de job (JOBID), pero con numero de tareas. Y se pueden suprimir de manera muy eficiente, son el solo JOBID:

```
$ qdel 53 54
jerome has deleted job 53
jerome has deleted job 54
$ qstat
$
```

En caso de ser mas especifico, se puede suprimir una sola tarea, por ejemplo la 112, utilizando el **qdel JOBID.112**

3.4.2. Un primer caso concreto

Se presenta un ejemplo de uso de Job Array: se supone que se tiene 100 archivos nombrados **muestras.X** con X variando de 1 hasta 100. Se debe de procesar por un programa **analyze** disponible desde el PATH del usuario. La salida sera llamada **resultados.X**. El script siguiente generara el Job Array adecuado:

```
!#/bin/bash

# Nombre del Job
#$ -N AnalysisviaArray

# Para asegurar que todo va bien
echo "valor de task id : $SGE_TASK_ID"

# opciÃ³n del programa  -i : input ; -o : output
analyze -i muestras.$SGE_TASK_ID -o resultados.$SGE_TASK_ID
```

3.4.3. Caso con Blast

Un job array es idÃ³neo cuando se trata de analizar numero largo de secuencia con Blast. Se propone un ejemplo con 100 secuencias fasta (o grupos de secuencia) de nucleÃ³tidos. Como se trata de secuencias de supuestos genes, la bÃºsqueda se harÃ¡ sobre la base de datos de proteÃnas **nr**., con **blastx**. Queremos buscar las Se ha generado la lista de estos archivos fasta en un archivo llamado **lista-secuencias.txt**.

El script aquÃ propuesto debe de contemplar como sacar el nombre del archivo de secuencias que le toca, y entregar el resultado con un nombre relacionado con su numero de tarea. Si se usarÃa un nombre de salida comÃºn, Â¡se sobrescribirÃan los resultados! una soluciÃ³n a este problema es lo siguiente:

```
#!/bin/bash

#$ -N ArrayBlast
```

```
# Tenemos 100 archivos a analizar
#$ -t 1-100

# Para el PATH
/share/apps/Profiles/share-profile.sh

# UbicaciÃ³n de las bases de datos formateadas
export BLASTDB=/scratch/BlastDB/

# Nombre de la lista
LISTA=$HOME/lista-secuencias.txt

OutFile=out-$SGE_TASK_ID.txt

# Extraer el nombre del archivo que lo toca:
# tarea 10: linea 10 !
FILE=$(awk "NR==$SGE_TASK_ID" $LISTA)

# Ejecutar la bÃºsqueda
blastx -query $FILE -db nr -evalue 0.001 -out $OutFile
```

3.5. Jobs con dependencia

El scheduler SGE permite definir dependencia entre jobs. Este permite, por ejemplo, determinar que un job B iniciara Ãºnicamente cuando el job A acabara. Es Ãºtil cuando hay dependencia de los resultados. La dependencia no esta limitada a un solo job, puede ser una lista de jobs.

Regresando al ejemplo anterior del Job Array con blastx, se podrÃa imaginar que la lista de las secuencias sea el resultado de un proceso anterior. El script siguiente, aunque muy simple, sirve como base para dividir las secuencias y genera la lista. En caso real, el flujo de los datos podrÃa ser mucho mÃ¡s complejo..

```
#!/bin/bash

#$ -n SplitFile

# el archivo original de 100 secuencias sera dividido en 100 archivos..
rm -rfv Split
mkdir Split
cd Split
seqretsplit -auto -sequence $HOME/archivoOriginal.fna

# Generamos la lista
cd
find $HOME/Split -name "*.fasta" > $HOME/lista-secuencias.txt
```

Este script se lanzara primero. Se tomara en cuenta su numero de JOBID. Y se mandara luego el jobs de Blast, con la condiciÃ³n de dependencia, utilizando la opciÃ³n **-hold_jid**.

```
$ qsub split.sh
```

```
Your job 63 ("SplitFile") has been submitted
$ qsub -hold_jid 63 blastarray.sh
Your job-array 64.1-100:1 ("ArrayBlast") has been submitted
$ qstat
job-ID  prior   name       user         state submit/start at     queue
---------------------------------------------------------------------------------------------
     63 0.55500 SplitFile  jerome       qw    02/17/2016 19:14:51
     64 0.00000 ArrayBlast jerome       hqw   02/17/2016 19:15:06
```

El job 64 tiene un estado **hqw**. El "h" significa que esta Hold, en espera de ser liberado. En este caso, sera cuando el job 63 acabe.

Es importante notar que si el job 63 falla o esta cancelado, el job 64 entrara para correr. Se debe de implementar un sistema de verificaciÃ³n dentro de los jobs con dependencia.

3.6. Jobs paralelos

Una de las ventajas de un cluster es que permite correr programas paralelos. Actualmente, los programas paralelos se desarrollan principalmente con las dos librerÃas siguientes (para los CPU):

1. Message Passing Interface (MPI)
2. OpenMP (Dentro de un mismo nodo)

Debido a la especificidad en el uso de cada de las librerÃas, se define 2 ambientes paralelos dentro de SGE, para agilizar su uso.

3.6.1. MPI

MPI es un estÃ¡ndar, y se basa en la nociÃ³n de canales de mensajes entre los diferentes cpus involucrados. Los canales de comunicaciÃ³n son prioritariamente la red IB, luego la red Ethernet, y tambiÃ©n canales internos al los CPUs modernos (intra-node). Cada cpu ejecuta el mismo cÃ³digo, y es la lÃ³gica del cÃ³digo que lo permite diferenciarse de los demÃ¡s con su numero de threads. Existe librerÃas para el lenguaje C, C++, Fortran entre otros.

Existen diferentes implementaciones de MPI. En el cluster esta instalado por defaut la implementaciÃ³n OpenMPI [2]. Los programas instalados en el cluster que utilizan MPI son compilados para esta implementaciÃ³n, al menos que los documentos en el cluster indican lo contrario.

Para poder utilizar los programas en MPI, se debe de usar un ambiente paralelo especifico en SGE. Este ambiente se encargara de iniciar los procesos y los canales para su comodidad. En este caso de MPI, se usara el ambiente **mpi**, gracias al uso de la opciÃ³n **-pe**, escogiendo una de las dos maneras siguientes:

- *-pe mpi 4* : se reservaran 4 cores para correr el job. Se escribe en linea de comando de qsub.
- *#$ -pe mpi 4* : se reservaran 4 cores para correr el job. Se escribe dentro del script.

Se proporciona al job una variable de ambiante, llamada **NSLOTS** que contiene el numero de cpus reservados. Esta variable *deberÃ¡* utilizarse para indicar a los programas del script cuantos cpus disponen.

AquÃ se presenta un script bÃ¡sico para correr este tipo de programas. Nota por favor la utilizaciÃ³n de la variable **NSLOTS**:

```
#!/bin/bash

#$ -N MpiJob

# 4 cores.
#$ -pe mpi 4

source /share/apps/Profiles/share-profile.sh

# Se usa mpirun para correr un programa MPI:
mpirun -np $NSLOTS program-mpi
```

3.6.2. OpenMP

OpenMP es una especificaciÃ³n que permite, dentro de programas en C (o otros lenguajes), definir partes de cÃ³digo que querremos que sea ejecutada en paralelo cuando se presenta la opciÃ³n. Esta especificaciÃ³n se permite Ãºnicamente dentro de la misma maquina (nodo), y no es capaz de usar cualquier red.

Como con el caso de MPI, se define un ambiente paralelo especifico para poder correr programas con OpenMP. Lo que realiza en particular el ambiente paralelo **thread**, es revisar que los cores pedidos por el job sean TODOS en el mismo nodo.

Con las caracterÃsticas actuales del cluster, no se podrÃ¡ pedir correr programas con OpenMP con mas de 64 cores. El scheduler les rechazara la peticiÃ³n, indicÃ¡ndole que lo que pide no existe.

Se proporciona al job una variable de ambiante, llamada **NSLOTS** que contiene el numero de cpus reservados. Esta variable *deberÃ¡* utilizarse para indicar a los programas del script cuantos cpus disponen.

Blast instalado en el cluster permite el uso de mas de un threads. El script siguiente muestra como correr un blast con 2 threads. Notan la utilizaciÃ³n de la variable **NSLOTS** :

```
#!/bin/bash

# Nombre del job
#$ -N  Blastn-NT

# En 2 threads.
#$ -pe thread 2
```

```
# Para obtener el PATH adecuado
source $HOME/.bashrc

module load programs/blast-2.9.0

# UbicaciÃ³n de las bases de datos formateadas.
export BLASTDB=/scratch/BlastDB/

OutFile="results.txt"

blastn -db nt -query nuc.fasta -num_threads $NSLOTS -out $OutFile
```

3.7. Opciones usuales

Existe opciones para modificar el comportamiento del job. Si se presenta las dos opciones (una en el script, otra en linea de comando), la opcion de la linea de comando tiene prioridad.

Les presentamos algunoas ociones mas comunes. El manual del comando qsub contiene mas informaciones sobre las opciones disponibles.

Table 3-1. Opciones SGE mas comunes

Linea de Comando	Script	Efecto
`-N MiJob`	`#$ -N MiJob`	`Nombre del job: MiJob`
`-j y`	`#$ -j y`	`Juntar STDOUT y STDERR`
`-l lenta`	`#$ -l lenta`	`Pedir la cola lenta.q`
`-l h_vmem=12G`	`#$ -l h_vmem=12G`	`Pedir 12 Gb por core`
`-cwd`	`#$ -cwd`	`Ejecucion en "pwd"`

Notes

1. Ver el manual de qsub para mas detalles

    2. http://gridscheduler.sourceforge.net/htmlman/htmlman1/qsub.html

2.

    3. http://www.openmpi.org

Chapter 4. Comandos Ãºtiles

4.1. Comandos remotos desde los nodos

En caso que usan los scratch locales de los nodos, puede ser interesante poder ver sus contenidos desde al mÃ¡ster del cluster, sin conectarse directamente a ellos. El cluster cuenta con una herramienta que lo permite. Es el comando **rocks run host compute**

Por ejemplo, para listar el contenido del scratch de cada nodo:

```
$ rocks run host compute command="ls -l /state/partition1/" collate=yes
compute-0-0: total 4194328
compute-0-0: drwxr-xr-jerome useruuab       4096 Jan 29 12:12 jerome
compute-0-0: drwx------.  2 root   root          16384 Mar  6  2015 lost+found
compute-0-0: -rw-r--r--.  1 root   root     4294967296 Mar 27  2015 test.0
compute-0-1: total 20
compute-0-1: drwxr-xr-x 9 jerome useruuab  4096 Jan 29 12:16 jerome
compute-0-1: drwx------ 2 root   root     16384 Mar  6  2015 lost+found
compute-0-2: total 20
compute-0-2: drwxr-xr-x 8 jerome useruuab  4096 Jan 29 12:14 jerome
compute-0-2: drwx------ 2 root   root     16384 Mar  6  2015 lost+found
compute-0-3: total 20
compute-0-3: drwxr-xr-x 8 jerome useruuab  4096 Jan 29 11:58 jerome
compute-0-3: drwx------ 2 root   root     16384 Mar  6  2015 lost+found
compute-0-4: total 20
compute-0-4: drwxr-xr-x 8 jerome useruuab  4096 Jan 29 12:16 jerome
compute-0-4: drwx------ 2 root   root     16384 Mar  6  2015 lost+found
```

Cada nodo tiene su propio espacio de scratch, es por eso que no todos tienen el mismo listado. La opciÃ³n **collate=yes** permite mostrar a cual nodo la linea corresponde.

Ahora, si se necesita copiar archivos de los nodos hasta su home, podemos usar el mismo comando para realizar esta tarea. Si los archivos a copiar se son en **/state/partition1/$USER/archivos*.txt**, pueden copiarlos asi:

```
$ rocks run host compute command="cp -v /state/partition1/$USER/archivo*.txt $HOME"
‘/state/partition1/jerome/archivo18.txt’ -> ‘/home/jerome/archivo18.txt’
‘/state/partition1/jerome/archivo33.txt’ -> ‘/home/jerome/archivo33.txt’
‘/state/partition1/jerome/archivo100.txt’ -> ‘/home/jerome/archivo100.txt’
‘/state/partition1/jerome/archivo99.txt’ -> ‘/home/jerome/archivo99.txt’
‘/state/partition1/jerome/archivo80.txt’ -> ‘/home/jerome/archivo80.txt’
‘/state/partition1/jerome/archivo81.txt’ -> ‘/home/jerome/archivo81.txt’
‘/state/partition1/jerome/archivo82.txt’ -> ‘/home/jerome/archivo82.txt’
```

4.2. Transferencia de archivos vÃa el NAT

Para los usuarios del cluster que acceden remotamente vÃa el *NAT*, existe el problema de la transferencia de los archivos entre su mÃ¡quina propia y el cluster. El servidor *NAT* no permite guardar archivos temporales, asi que

se debe de utilizar la tÃ©cnica de un **tÃºnel** con ssh. La base de esta tÃ©cnica es a base de 2 terminales. En el primero nos conectaremos al NAT creando un tÃºnel hacia el cluster. En el segundo terminal, copiaremos los archivos utilizando este tÃºnel. Todo ello se realiza desde su mÃ¡quina personal, suponeniendo que no tiene una IP publica y accesible.

Presentamos un ejemplo de utilizaciÃ³n de este mÃ©todo, para un usuario que tenga una cuenta "*usuario*" tanto en el servidor *NAT* que en *Teopanzolco*.

En el primer terminal, vamos a conecarnos al NAT crando al mismo tiempo un tÃºnel. Este tÃºnel sera con un nÃºmero de puerto, que debe de ser mayor a 1024. En nuestro exemplo, proponemos **6789**. Se pedira la contraseÃ±a del usuario para el NAT:

```
$ ssh -T -L 6789:teopanzolco.ibt.unam.mx:22 usuario@nat.ibt.unam.mx
usuario@nat.ibt.unam.mx's password: La_contraseÃ±a_en_el_NAT

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Could not chdir to home directory /home/usuario: No such file or directory
```

Notaran que con la opciÃ³n **-T**. no dara ningÃºn prompt el comando ssh.

Una vez que la conecciÃ³n esta establecida en el primer terminal, vamos a utilizar el tÃºnel creado en el puerto 6789 para enviar o recibir archivos directamente desde el cluster. Si quieremos copiar el archivo **reads1.fastq.gz** desde la mÃ¡quina personal hacia teopanzolco, dejandole el mismo nombre, teclearemos el comando siguiente. El mensage de aviso de la autenticidad aparecera la primera vez que se realiza esta conecciÃ³n. Se entra entonces la contraseÃ±a del usuario en el cluster, esta vez!

```
$ scp -P 6789 reads1.fastq.gz usuario@localhost:
The authenticity of host '[localhost]:6789 ([::1]:6789)' can't be established.
ECDSA key fingerprint is SHA256:UinWwEhlAZSen26tBdl4YpiOqvp1OecMmYYvZv28LIg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:6789' (ECDSA) to the list of known hosts.
Password: La_contraseÃ±a_en_teopamzolco
reads1.fastq.gz                                     100%   25KB   5.5MB/s   00:00
```

Si queremos copiar un archivo **resultado.txt** desde el cluster hacia nuestra mÃ¡quina personal, quedandonos con el mismo nombre, teclearemos el comando siguiente:

```
$ scp -P 6789 jerome@localhost:resultado.txt  .
Password: La_contraseÃ±a_en_teopamzolco
resultado.txt                                      100%   25KB   5.8MB/s   00:00
```

Cuando terminamos con las transferencias, no olvidemos cancelar el tÃºnel, simplement con [CTRL]-C en el primer terminal.

Chapter 5. Copyrights

5.1. MÃ¡s informaciones

Todos los programas usados en el cluster son bajo licencias libres tipo GPL

Cualquier comentario o peticiÃ³n a este documento se debe de mandar al correo **clustersupport** arobas **ibt.unam.mx**.