INSTITUTO DE INVESTIGACIONES DE LA AMAZONIA PERUANA - IIAP

PROGRAMA DE INVESTIGACIÓN EN INFORMACIÓN DE LA BIODIVERSIDAD AMAZÓNICA - BIOINFO

CENTRO DE ALTO RENDIMIENTO COMPUTACIONAL DE LA AMAZONIA PERUANA

MANUAL SOBRE EL USO DE LA SUPERCOMPUTADORA "MANATI" DEL IIAP

Por:

Américo Sánchez C.

Isaac Ocampo Y.

Rodolfo Cárdenas V.

FEBRERO 2018

Sumario

I.EL SUPERCOMPUTADOR MANATÍ 2

II.PROCEDIMIENTO PARA SOLICITAR ACCESO A LA CLUSTER HPC “MANATI” DEL IIAP ... 5

III.INSTRUCCIONES PARA EL ACCESO REMOTO A LA CLÚSTER HPC “MANATI” DEL IIAP . 8

3.1.Requisitos de software:..... 8

3.2.Conexión remota por SSH 8

3.3.Transferencia de archivos 11

IV.EJECUCIÓN DE PROGRAMAS EN LA COLA DE TRABAJO PBS..... 13

4.1 Ejecutar programa en la cola de trabajo PBS 14

4.1.1 Requisitos 14

4.1.3 Creación de bash script para PBS..... 15

ANEXO 1 – COMANDOS BÁSICOS DE LINUX..... 17

ANEXO 2 – SOFTWARE Y HERRAMIENTAS USADOS EN SUPERCOMPUTACIÓN 22

ANEXO 3 – MANUAL DEL ADMINISTRADOR DE COLAS DE TRABAJO 25

I. EL SUPERCOMPUTADOR MANATÍ

A finales de 2016, el Instituto de Investigaciones de la Amazonia Peruana – IIAP, implementó en su sede principal (Iquitos) un Sistema de Alto Rendimiento Computacional tipo Clúster HPC[1], en el marco del proyecto “Fortalecimiento de la infraestructura tecnológica para procesos de investigación del Instituto de Investigaciones de la Amazonia Peruana” financiado por CIENCIACTIVA de CONCYTEC.

¿Qué es Manatí y cuál es su finalidad?

El Supercomputador Manatí es un sistema computacional de alto rendimiento diseñado para apoyar investigaciones científicas y tecnológicas que tengan necesidades de procesamiento de grandes volúmenes de información (gráficos y numéricos), cálculos matemáticos, simulación, modelamiento y otras operaciones computacionales que requieren de grandes capacidades de procesamiento de cómputo para obtener ventajas con respecto al tiempo de respuesta, eficiencia y resultados más confiables en comparación a computadoras convencionales.

La supercomputadora del IIAP denominada MANATI tiene una arquitectura tipo Cluster, compuesta por 10 nodos (servidores o computadoras con grandes capacidades) distribuidos de la siguiente manera, un nodo coordinador y 9 nodos de procesamiento (6 gráficos y 3 numéricos):

- 1 Nodo Principal: CPU Intel Xeon con 56 núcleos, 64 GB de RAM
- 6 Nodos Procesamiento Grafico: CPU Intel Xeon con 28 núcleos, GPU NVIDIA TESLA K80 con 4992 Cuda cores, 64 GB de RAM
- 3 Nodos Procesamiento Grafico: CPU Intel Xeon con 28 núcleos, 64 GB de RAM

1 High Performance Computing, computación de alto rendimiento

- Sistema de almacenamiento de 114 TB.

El sistema operativo de la supercomputadora Manatí es Linux Centos 7.

¿Qué tipos de servicios brinda Manatí?

Los tipos de servicios que brinda el Supercomputador Manatí son:

- ✓ Procesamiento de información numérica (ejecución de aplicaciones, algoritmos, modelos matemáticos y paralelización, etc).
- ✓ Procesamiento gráfico (pre-procesamiento y procesamiento de imágenes espaciales).

¿Cuál es la población objetivo de los servicios que presta?

El Supercomputador Manatí es una herramienta de última generación que se encuentra al servicio de la ciencia nacional y en especial de toda aquella ciencia tendiente a apoyar la sostenibilidad de la Amazonía peruana.

De acuerdo con ello pueden acceder a los servicios del Supercomputador Manatí todo aquel científico nacional que desee apoyarse en la herramienta para ampliar las fronteras del conocimiento y/o estudiar problemáticas amazónicas.

A la fecha se ha facilitado el acceso a diversos investigadores, docentes y estudiantes de diversas universidades e instituciones nacionales. En la siguiente figura se presenta un gráfico resumen sobre las cantidades de trabajos de investigación que se encuentran en marcha al interior del Supercomputador Manatí.

Figura N.º 1 Grafico sobre las cantidades de trabajos de investigación de universidades e instituciones que aprovechan las capacidades de Manatí

¿Cuáles son los requisitos para acceder al uso de Manatí?

Los requerimientos mínimos para el uso de MANATI son:

- **Conocimiento básico de Linux.** En el Anexo 1 se presenta las descripciones de los principales comandos del sistema operativo Linux.
- **Conocimiento básico de software y/o programacion para HPC.** En el Anexo 2 se presenta las descripciones de los principales software y herramientas usados en supercomputación bajo sistema operativo Linux.
- **Conocimiento de manejo y acceso a conexiones remotas vía protocolo SSH.** Este aspecto se desarrolla en el punto III. INSTRUCCIONES PARA EL ACCESO REMOTO A LA CLÚSTER HPC “MANATI” DEL IIAP del presente manual.

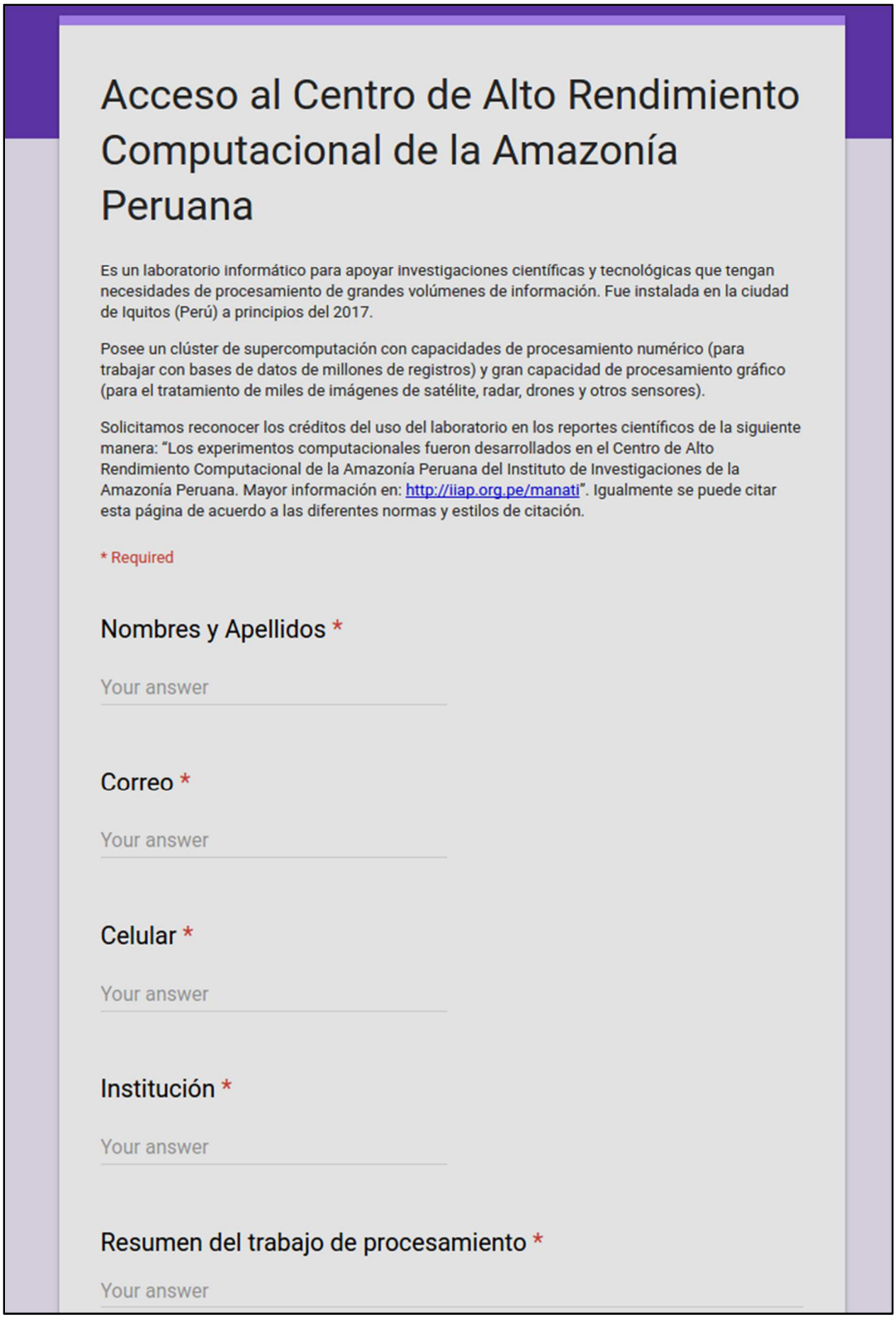
II. PROCEDIMIENTO PARA SOLICITAR ACCESO A LA CLUSTER HPC “MANATI” DEL IIAP

Para solicitar acceso a la cluster HPC Manatí es necesario completar el formulario de registro disponible en la siguiente dirección: https://goo.gl/v83Yiy

En las figuras 2 y 3 se puede apreciar los campos necesarios para completar el formulario de registro.

El equipo del Centro de Alto Rendimiento Computacional de la Amazonía Peruana (CARCAMP) perteneciente al Programa BIOINFO validará los datos y se comunicará con el solicitante en caso de haber dudas.

Posteriormente, al registrarse para el acceso y obtener el usuario y contraseña para el ingreso remoto, los usuarios se comprometen a reconocer los créditos del uso del laboratorio en los reportes científicos de la siguiente manera: *“Los experimentos computacionales fueron desarrollados en el Centro de Alto Rendimiento Computacional de la Amazonía Peruana del Instituto de Investigaciones de la Amazonía Peruana. Mayor información en: http://iiap.org.pe/manati”.*



Acceso al Centro de Alto Rendimiento Computacional de la Amazonía Peruana

Es un laboratorio informático para apoyar investigaciones científicas y tecnológicas que tengan necesidades de procesamiento de grandes volúmenes de información. Fue instalada en la ciudad de Iquitos (Perú) a principios del 2017.

Posee un clúster de supercomputación con capacidades de procesamiento numérico (para trabajar con bases de datos de millones de registros) y gran capacidad de procesamiento gráfico (para el tratamiento de miles de imágenes de satélite, radar, drones y otros sensores).

Solicitamos reconocer los créditos del uso del laboratorio en los reportes científicos de la siguiente manera: "Los experimentos computacionales fueron desarrollados en el Centro de Alto Rendimiento Computacional de la Amazonía Peruana del Instituto de Investigaciones de la Amazonía Peruana. Mayor información en: http://iiap.org.pe/manati". Igualmente se puede citar esta página de acuerdo a las diferentes normas y estilos de citación.

* Required

Nombres y Apellidos *

Your answer

Correo *

Your answer

Celular *

Your answer

Institución *

Your answer

Resumen del trabajo de procesamiento *

Your answer

Figura N.º 2 Primera parte del formulario de solicitud de acceso al Supercomputadora Manatí

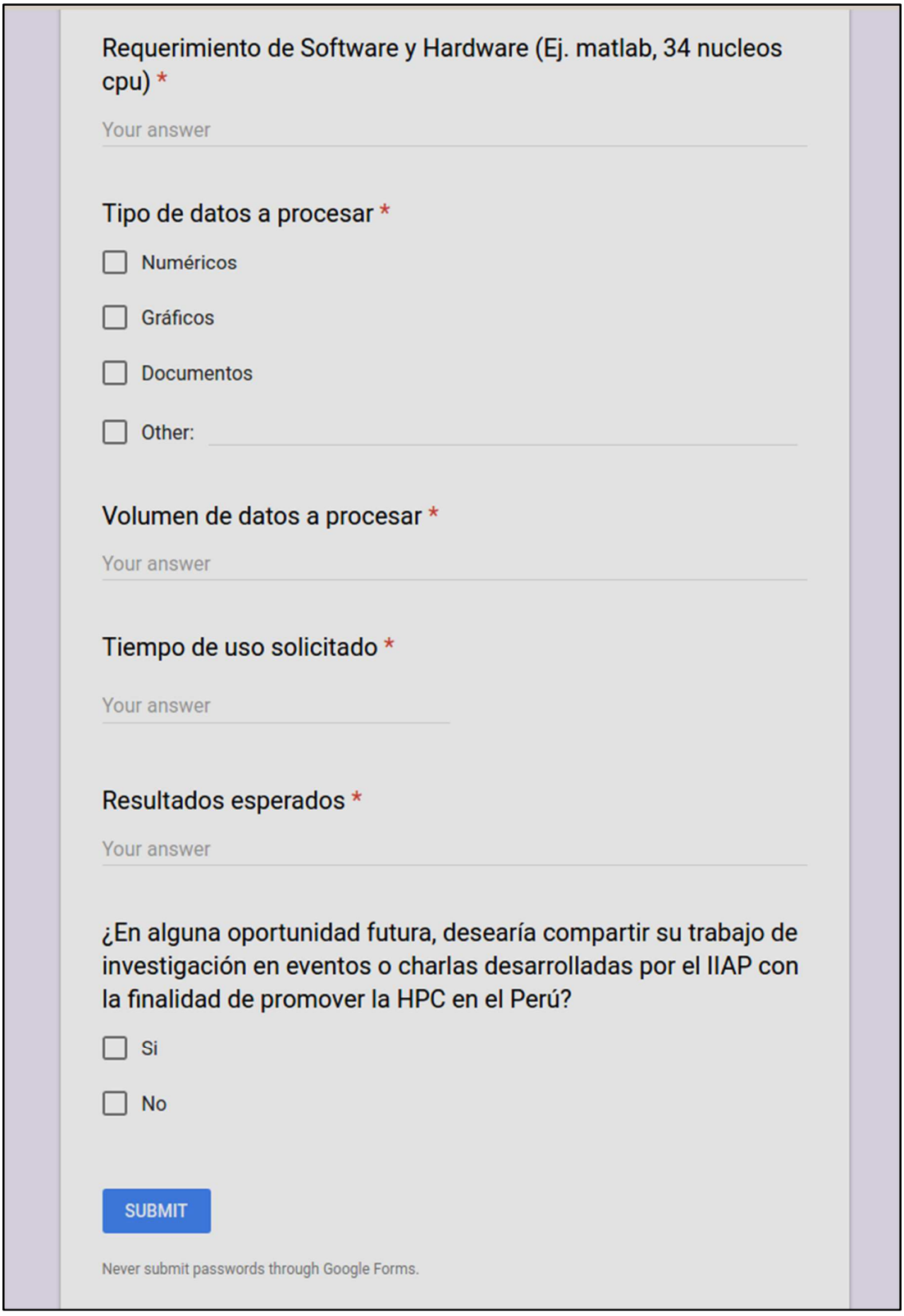


Figura N.º 3. Segunda parte del formulario de solicitud de acceso al Supercomputadora Manatí

III. INSTRUCCIONES PARA EL ACCESO REMOTO A LA CLÚSTER HPC “MANATI” DEL IIAP

Como mencionamos anteriormente, la supercomputadora del IIAP tiene un nodo coordinador, por medio del cual se hace uso de los nodos de procesamiento. Para ello es necesario conectarse por medio de una conexión remota segura, utilizando el protocolo de comunicación SSH, como describimos a continuación:

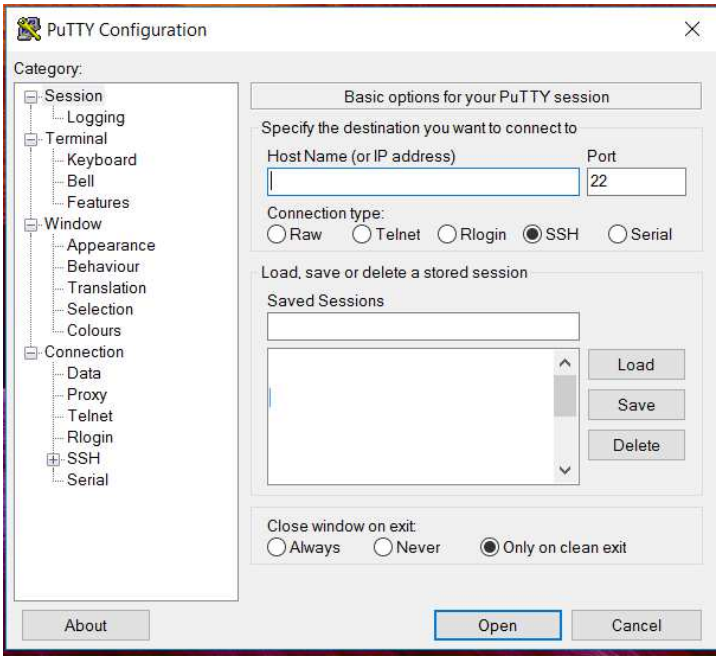
3.1. **Requisitos de software:**

a. Tener instalado el programa Putty, en este caso para usuarios que utilizan Windows, **https://the.earth.li/~sgtatham/putty/latest/w32/putty.exe**

b. Tener instalado el programa WinSCP para en este caso para usuarios que utilizan Windows , **https://winscp.net/eng/download.php**

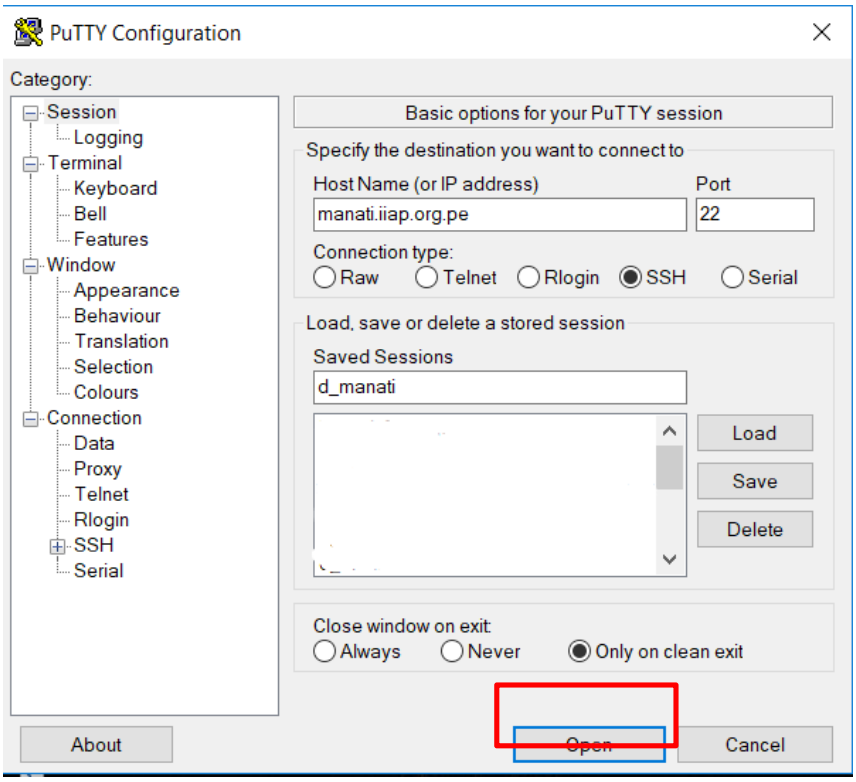
3.2. **Conexión remota por SSH**

a. Abrir el programa Putty:

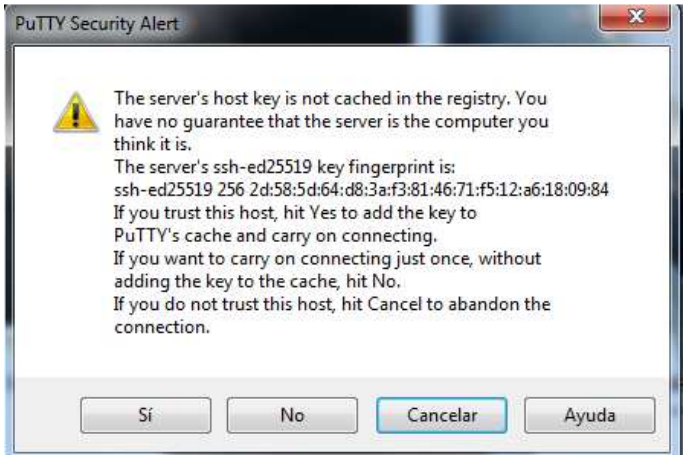


b. En el campo “Host Name (or IP address)” ingresar el dominio o host asignada a la supercomputadora del IIAP: **manati.iiap.org.pe**

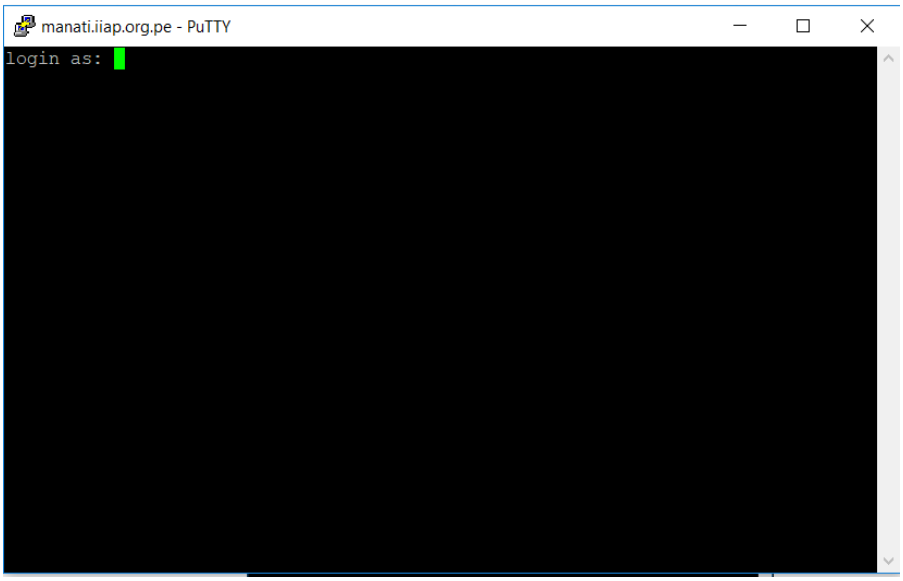
c. En el campo Port ingresar el puerto 22 y hacer click en Open:



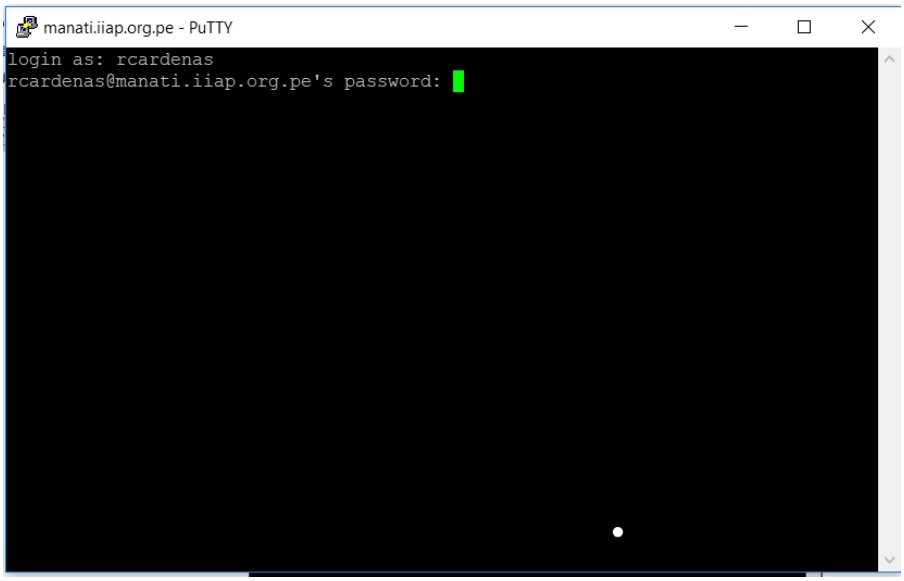
d. La primera vez que se ingresa aparece una ventana como la siguiente, hacer click en el botón **Sí**.



e. Ingresar el usuario de acceso asignado, pulsar la tecla **Enter**.



f. Ingrese la contraseña y pulse al final la tecla **Enter**

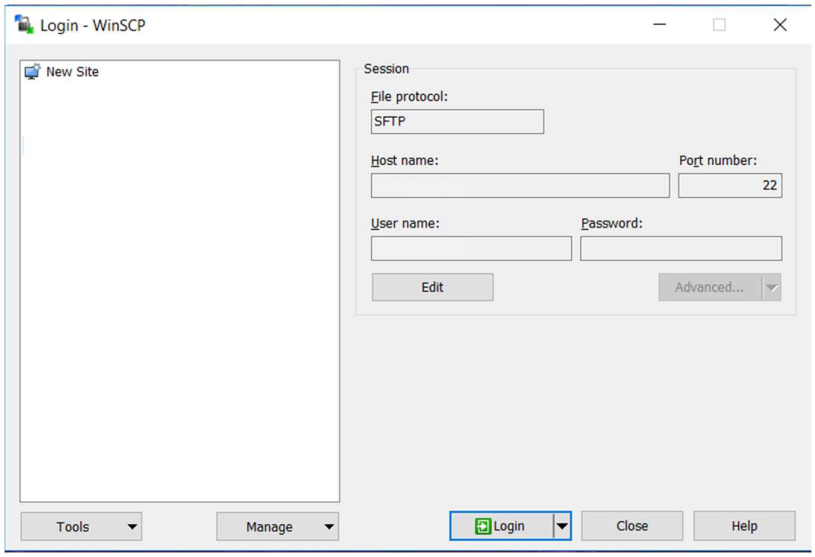


Nota: (cambio de contraseña): Si accede por primera vez, el sistema le pedirá volver a ingresar la contraseña actual (asignada por el administrador) y a continuación le pedirá ingresar dos veces una

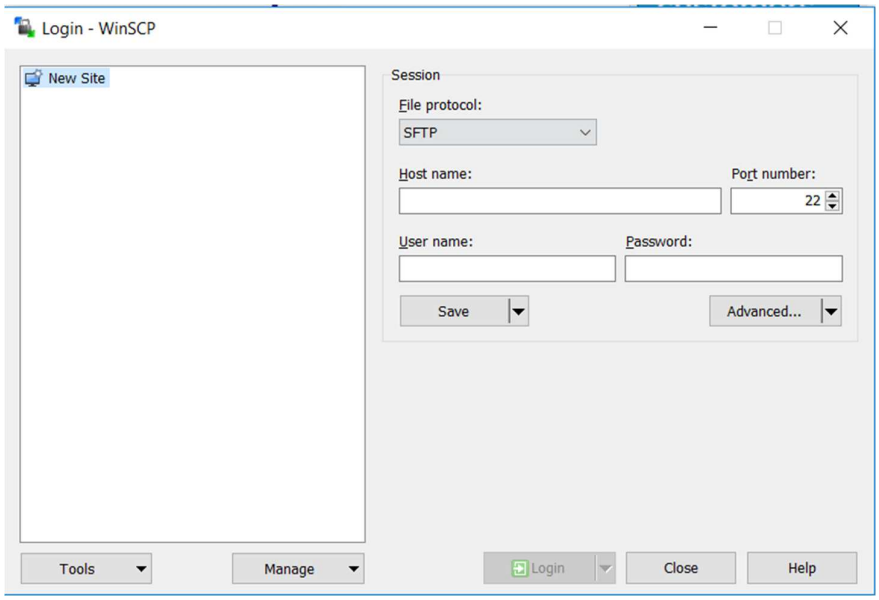
nueva contraseña. Se cerrará putty y tendrá que volver a acceder con su nueva contraseña.

3.3. **Transferencia de archivos**

a. Abrir el programa WinSCP:



b. Seleccionar opción “New Site” para crear una nueva conexión a la supercomputadora para transferir los archivos, se muestra la siguiente interfaz:



c. Llenar los siguientes campos:

i. Host name: **manati.iiap.org.pe**

ii. Port number: **22**

iii. User name: usuario

iv. Password: contraseña

Presionar el botón “**Login**”.

Nota: Para guardar los datos de conexión hacer click en botón “**Save**”

d. Se muestra la siguiente interfaz, en la cual al lado izquierdo (1) se encuentran los archivos de la maquina local y al lado derecho (2) se encuentran los archivos del usuario logueado en la supercomputadora.
Nota: Para transferir archivos arrastre de la sección 1 a la sección 2.

Una vez se haya ingresado remotamente al Supercomputador Manatí es necesario el uso del gestor de colas o tareas que administra los nodos: el software PBS (Portable Batch System). En la siguiente sección se proporcionarán los pasos iniciales para comenzar a utilizar esta herramienta clave para el procesamiento de información en las investigaciones científicas.

IV. EJECUCIÓN DE PROGRAMAS EN LA COLA DE TRABAJO PBS

Es importante conocer la arquitectura de la supercomputadora del IIAP para poder aprovecharla al máximo, y además los programas de terceros, programas compilados o desarrollados por el usuario deben utilizar al máximo los recursos del computador (uso de forma paralela de los procesadores CPU o GPU). El nodo coordinador sólo es utilizado para loguearse, subir los datos, instalar o compilar los programas de terceros o desarrollados por los usuarios y realizar algunas configuraciones necesarias, la ejecución de estos se realiza en los nodos de procesamiento. Como se puede observar en la

siguiente imagen, para lograr esto es necesario utilizar el gestor de tareas PBS que describiremos más adelante. ***Cabe recalcar que si los usuarios ejecutan sus programas en el nodo principal, esto afectará al desempeño del nodo ocasionando que se ponga lento o limite el acceso a otros usuarios.***

Arquitectura de la supercomputadora MANATI del IIAP.

Los usuarios de una supercomputadora se dividen en 3 grupos, el primero es el administrador(es); el segundo grupo, investigadores afines a las ciencias de la computación que desarrollan algoritmos o programas; y el tercer grupo, investigadores de diferentes áreas del conocimiento que hacen uso de programas desarrollados por terceros para procesar su información. En este manual nos centraremos en estos dos últimos grupos.

4.1 Ejecutar programa en la cola de trabajo PBS

4.1.1 Requisitos

a. Loguearse en la supercomputadora
b. Si eres usuario que pertenece al tercer grupo, debe estar instalado el programa o software a utilizar, en caso de no estar instalado comunicarse con el administrador al correo rcardenasv@iiap.org.pe. Para este ejemplo utilizaremos el Programa “R”.
c. En el caso de los usuarios que pertenecen al segundo grupo, deben compilar el código fuente de su algoritmo o programa en el nodo principal para generar su ejecutable, en caso de no existir los compiladores necesarios comunicarse con el administrador al correo rcardenasv@iiap.org.pe.

4.1.2 Creación de script en R.

a. Crear un archivo scrip con la extensión .R, en la cual se indicará la secuencias de comandos que debe ejecutar R:

```
nano test.serial.R
```

b. Se escribe el siguiente código:

```
for (i in 1:10) {

 hola=paste("Hola probando R en i=",i,sep=" ")
 print(hola)
}
```

c. Se guarda el archivo con la tecla **"Ctrl"+"x",** luego se presiona la tecla **"y"** finalmente la tecla **enter**.

4.1.3 Creación de bash script para PBS

d. Crear un archivo **BASH** con la extensión **.sh**, en la cual se indicará las instrucciones del gestor de colas PBS para ejecutar su programa en los nodos de procesamiento. Utilizar el siguiente comando:

```
nano pbs_test_r.sh
```

e. Se ingresa el siguiente código (no confundir el número 1 con la letra l):

```
#!/bin/sh
#PBS -N test_R  //nombre tarea
#PBS -l nodes=1:ppn=1 //cantidad de nodos y núcleos

cd $PBS_O_WORKDIR
#Comandos utilizados para ejecutar su programa
/opt/shared/R-3.4.2/bin/R CMD BATCH test.serial.R
```

En caso de trabajar con GPUS la tercera línea debe ser:

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

Nota: Revisar el manual de PBS para una mayor comprensión de los comandos de PBS (Anexo 3).

f. Guardamos el archivo con la tecla **“Ctrl”+”x”**, luego se presiona la tecla **“y”** finalmente la tecla **Enter**.
g. Ejecutamos la tarea con el comando:

```
qsub pbs_test_r.sh
```

h. Esperamos a que la tarea termine, eso se puede verificar con el comando:

```
qstat
```

En la columna estado la tarea debe estar en E o C, que quiere decir que se terminó de ejecutar la tarea.

i. PBS genera dos archivos con extensión .eXXX[2] y .oXXX, conteniendo mensajes de error y salidas de consola respectivamente, para mostrar el contenido puede hacer uso del comando:

```
cat test_R.o41538
```

Pero en este caso los resultados se guardan en el archivo con extensión .Rout generado por “R”. Para ver los datos use el comando:

```
cat test.serial.Rout
```

2 Siendo XXX el número que figura en la primera columna del resultado del comando ejecutado en el punto e. (ID del Job)

ANEXO 1 – COMANDOS BÁSICOS DE LINUX

A continuación presentamos los comandos más utilizados en los sistemas operativos en Linux de acuerdo con la web Open Webinars y su guia definitiva para emprender a usar la terminal de Linux

Navegando por los directorios:

pwd: “Print working directory” (Muestra el directorio de trabajo), nos mostrará la ruta en la que nos encontramos actualmente. Muy útil si hemos estado saltando de subcarpeta en subcarpeta y el prompt nos muestra sólo una ruta abreviada.

ls: Nos muestra una lista con el contenido del directorio actual (o el que le pasemos como argumento, por ejemplo: “ls /home/usuario”).

ls –l: Muestra una lista del contenido del directorio añadiendo información adicional de los ficheros o carpetas, como permisos, fecha y hora de creación o modificación, etc…

ls –a: Muestra una lista de todos los ficheros del directorio, incluyendo los ficheros o carpetas ocultos.

cd: nos lleva al directorio raíz.

cd.. : Subiremos un nivel en el árbol de directorios. Si por ejemplo nos encontramos en /home/usuario, con este comando nos iremos a /home.

Examinando archivos:

file: determina el tipo de un archivo.

cat: muestra el contenido de un archivo

less: muestra el contenido de un archivo, y lo va paginando en caso de que sea necesario por ser muy extenso.

Manipulando archivos y directorios:

cp: Copia un fichero o directorio.

cp –i: Copia un fichero o directorio y pregunta antes de sobrescribir si se diese el caso.

cp –r: Copia un directorio con todo su contenido.

mv: Mueve o renombra un fichero o directorio. En la terminal de Linux, en lugar de renombrar un fichero mediante un comando exclusivo, utilizamos el mismo que para mover archivos o directorios, y lo que hacemos para renombrarlo es moverlo a la misma ruta donde se encuentra y cambiarle el nombre.

mv –i: Mueve o renombra un fichero o directorio preguntando antes de sobrescribir si se diese el caso.

mkdir: Crea un directorio.

rmdir: Elimina un directorio vacío.

rm: Elimina un fichero.

rm –r: Elimina un directorio y todo su contenido.

rm –i: Elimina un fichero solicitando confirmación. Es muy recomendable usarlo con la opción –r para poder usarlo con directorios evitando problemas.

Atajos de teclado:

Ctrl + Alt + Fn (1-6): Para abrir terminales a pantalla completa. Requiere autentificación con usuario y contraseña. Cambiaremos a la terminal correspondiente a la tecla **Fn** que pulsemos, correspondiendo ésta al orden en el que las abrimos. En Linux podemos tener múltiples terminales funcionando simultáneamente, por defecto controlaremos hasta 6 con esta combinación de teclas. Por ejemplo **Ctrl + Alt + F**1 nos lleva a la primera terminal abierta.

tty: Con este comando, en el caso de tener varias terminales abiertas, nos dirá en cuál nos encontramos.

Ctrl + Alt + F7: Nos devolverá al entorno gráfico (si usábamos alguno).

Shift (Mayus) + RePág: Realizaremos scroll hacia arriba en la terminal.

Shift (Mayus) + AvPág: Realizaremos scroll hacia abajo en la terminal.

Tab (Tabulador): Completará el comando, nombre de fichero o directorio que estemos escribiendo. En caso de múltiples coincidencias, con una doble pulsación de esta tecla obtendremos todos los resultados posibles encontrados en el directorio o sistema.

Flecha de dirección 'arriba' y /o 'abajo': Nos moveremos por el historial compuesto por los últimos comandos usados.

Ctrl + r: Buscará en el historial el último comando usado según lo que vayamos escribiendo. Por ejemplo utilizamos el comando clear hace unas horas, si pulsamos Ctrl + r y escribimos cl nos mostrará el comando clear, puesto que lo ha buscado en el historial y es el más reciente que coincide.

Ctrl + c: Interrumpe cualquier proceso en ejecución de forma inmediata y nos devuelve al prompt.

Ctrl + z: Envía el proceso actual a segundo plano. Para recuperarlo sólo tendremos que escribir fg y pulsar Intro.

Ctrl + d: Cierra la sesión de la terminal en la que nos encontramos. Si estamos usando una interfaz gráfica en la que hemos abierto una terminal, ésta sólo se cerrará.

Ctrl + w: Elimina la palabra anterior a la posición del cursor.

Ctrl + k: Corta todo aquello que se encuentra entre la situación del cursor y el final de la línea.

Ctrl + u: Corta la línea en la que nos encontramos al completo.

Ctrl + y: Pega el contenido copiado o cortado con Ctrl + u o Ctrl + k.

!!: Repetirá el último comando usado.

Comandos de Información del Sistema:

date: Muestra la fecha y hora al completo.

cal: Muestra el calendario del mes en curso.

whoami: Muestra el nombre del usuario con el que estamos trabajando.

w: Muestra qué usuarios están conectados actualmente.

uptime: Muestra el tiempo que lleva encendido el sistema, y cuántos usuarios lo han usado.

uname –a: Ofrece información del Kernel del sistema.

cat /proc/cpuinfo: Muestra información del Microprocesador

cat /proc/meminfo: Muestra información de la memoria del equipo.

free: Muestra la cantidad de memoria total, usada y libre, así como el espacio en la unidad de intercambio.

Comandos de administración:

adduser xxxxxx: Donde sustituiremos las **x** por el nombre del usuario que queramos añadir.

passwd xxxxxxx: Donde **xxxxxxx** será el nombre del usuario al que queramos cambiar la contraseña. Necesitaremos conocer la contraseña ya establecida si queremos cambiarla.

su: Inicia sesión como superusuario o root desde la sesión actual.

exit: Cierra la sesión del superusuario o root, volviendo al usuario desde la que se inició.

Comandos de proceso:

ps: Muestra los procesos que se encuentran activos en el sistema actualmente.

top: Muestra todos los procesos en funcionamiento.

kill “pid” (process id): Detiene el proceso asignado al **pid** que muestra la salida del comando **ps**.

bg: Muestra todos los procesos pausados o en segundo plano (recordamos que **Ctrl + z** establecía procesos en segundo plano).

fg: Trae de vuelta el proceso más reciente puesto en segundo plano.

Comandos de actualización:

yum install xxxxxx: Instalador de paquetería para gestores de paquetes rpm.

rpm –i xxxxxx.rpm: Instalador de paquetería para gestores de paquetes rpm.

Comandos de inicio y apagado:

halt: Detiene todos los procesos y apaga el equipo.

shutdown: Programa el apagado del sistema en 1 minuto.

shutdown –r X: Programa el reinicio del sistema, donde X será el número de minutos en el que se reiniciará el equipo.

shutdown –h now: Apaga el equipo saltándose el minuto de espera programado.

shutdown –r now: Reinicia el equipo saltándose la espera programada.

reboot: Reinicia el sistema de la misma forma que el comando anterior.

init 0: Apaga el sistema.

init 6: Reinicia el sistema.

startx: Inicia la interfaz gráfica si ésta se encuentra instalada en el sistema.

Comandos de Red:

ifconfig: Lista las direcciones **IP** de todos los dispositivos del equipo.

ping xxxx: Manda una señal que deberá ser devuelta por el equipo **xxxx** para comprobar si se encuentra en línea o no.

whois xxxxx: Obtiene información acerca de un dominio **xxxxx**, como por ejemplo www.google.com

wget xxxx: Descargará el archivo **xxxx**. Deberemos proporcionarle una dirección completa como por ejemplo: https://direccionip/carpeta/subcarpeta/archivo.file

Comandos de comandos:

man xxxxx: Muestra el manual de uso o configuración del programa xxxxx.

man –k xxxxx: Muestra las páginas de manual que contengan la palabra xxxxx.

apropos xxxxx: Lista las páginas de manual que tratan acerca del comando xxxxx.

whereis xxxxx: Muestra la localización más probable para el programa xxxxxx.

ANEXO 2 – SOFTWARE Y HERRAMIENTAS USADOS EN SUPERCOMPUTACIÓN

De acuerdo con el libro Introducción a la Supercomputación, autoría del IIAP, a continuación se presenta un listado de: software para desarrollo de aplicaciones y cálculo numérico, APIs, librerías y de temáticas especializadas.

1. Software para desarrollo de aplicaciones y calculo numérico en supercomputadoras

N°	Nombre	Año de creación	Usos principales
1	C	1970	Se ha utilizado para el desarrollo de muy diversas aplicaciones: sistemas operativos, hojas de cálculo, gestores de bases de datos.
2	C++	1983	Sistemas Operativos, drivers de dispositivos, gestores de paquetes y programas de configuración, aplicaciones gráficas de usuario, entre otros
3	Fortran	1957	Se utiliza principalmente para aplicaciones científicas y el análisis numérico
4	Python	1991	Desarrollar software para aplicaciones científicas, para comunicaciones de red, para aplicaciones de escritorio con interfaz gráfica de usuario (GUI), para crear juegos, para smartphones y para aplicaciones web.
5	R	1993	Investigación por la comunidad estadística, siendo además muy popular en el campo de la minería de datos, la investigación biomédica, la bioinformática y las matemáticas financieras.

6	Matlab	1984	Análisis de datos, modelado y diseño de sistemas DSP, modelado y control de sistemas de potencia y guiado
7	Octave	1988	Está orientado al análisis y cálculo numérico
8	Scilab	1994	Matemáticas y simulación, funciones de gráficos para visualizar 2D y 3D, herramientas para realizar análisis de datos y modelización, Xcos: simulador por diagramas en bloque de sistemas dinámicos híbridos, simulador de sistemas de modelado mecánico, ciruitos hidráulicos, entre otros.

2. APIs y librerías

N°	Nombre	Año de creación	Usos principales
1	Apache Hadoop	2011	Procesamiento distribuido de grandes conjuntos de datos a través de grupos de ordenadores
2	Cuda	2007	Procesamiento de vídeo e imágenes, datos de biología y química computacional, simulación de la dinámica de fluidos, reconstrucción de imágenes de TC, análisis sísmico o trazado de rayos, entre otras.
3	MPI	1994	Es un mecanismo para la programación paralela por paso de mensajes. Por ello, no es necesario que todos los procesos MPI ejecuten en el mismo nodo
4	OpenCL	2008	Permite implicar en los cálculos de una tarea a todos los núcleos del procesador central o toda la capacidad de computación de la unidad de procesamiento gráfico, cosa que, a fin de cuentas, reduce significativamente el tiempo de ejecución de un programa. Por eso, el uso de OpenCL es muy útil en las tareas relacionadas con computaciones muy laboriosas o que gastan muchos recursos.

5	OpenMP	1997	Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join.
6	TBB	2006	Usado en predicción meteorológica numérica, oceanografía y astrofísica, análisis de elementos finitos, IA y automatización, ingeniería genética

3. Software especializado

N°	Nombre	Año de Creación	Usos Principales
1	Ambertools 16	2002	Simulación de campos de fuerza de dinámica molecular de biomoléculas
2	Gaussian	1970	Química Computacional
3	ORCA	1999	Química cuántica con énfasis específico en las propiedades espectroscópicas de las moléculas open-shell.
4	AutoDock	1990	Simulación de modelado molecular, especialmente para acoplamiento de proteína-ligando
5	Dock 6	1980	Predecir modos de unión de los pequeños complejos de molécula de proteínas, bases de datos de búsqueda de ligandos para los compuestos que se unen a una proteína particular, bases de datos de búsqueda de ligandos para compuestos que se unen dianas de ácido nucleico, examinar las posibles orientaciones de unión de la proteína-proteína y proteína de ADN, ayudar a guiar los esfuerzos sintéticos mediante el examen de pequeñas moléculas que se derivatizan computacionalmente

ANEXO 3 – MANUAL DEL ADMINISTRADOR DE COLAS DE TRABAJO (DESARROLLADO POR TECNOSYS S.A)

Guía de uso

Administrador de colas de trabajo

Daniel Benavides S.

Indice

Indice....2
Comandos....4
qsub....4
qdel....4
qstat....4
qalter....6
pbsnodes....7
Jobs....9
Distintos tipos de lanzamiento de jobs....11
Lanzar un job interactivo....11
Lanzar un job con dependencias....11
Trabajo con variable....11
Ejemplos de script....13
Administración del servidor....16
Ejecución de qmgr mediante su propia shell....16
Ejecución mediante la shell del sistema operativo....17
Manejo de colas (queue)....17

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.

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.

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]
```

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 ~]$
```

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

```
#!/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
```

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***).

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

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***".

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
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