Centro de Computación de Alto Rendimiento

CCAR_UNED

BRUNO – Manual de usuario

Versión 1.3

#SOMOS2030

ccar.uned.es

UNED

Centro de Computación de Alto Rendimiento

CLÚSTER BRUNO – Manual de Usuario (v.1.3)

Centro de Computación de Alto Rendimiento CCAR

Abril 2022

Sumario

Introducción

El Centro de Computación de Alto Rendimiento de la UNED (**CCAR**) tiene como misión proveer de la infraestructura y los recursos necesarios para investigadores que requieran el uso de herramientas de computación de alto rendimiento (HPC).

Su uso está destinado exclusivamente a labores de investigación y explotación de recursos computacionales en el ámbito de las Ciencias. El **CCAR** ofrece servicios de asesoría, adquisición de equipos e infraestructuras, soporte y mantenimiento de hardware y software de altas prestaciones.

Estructura

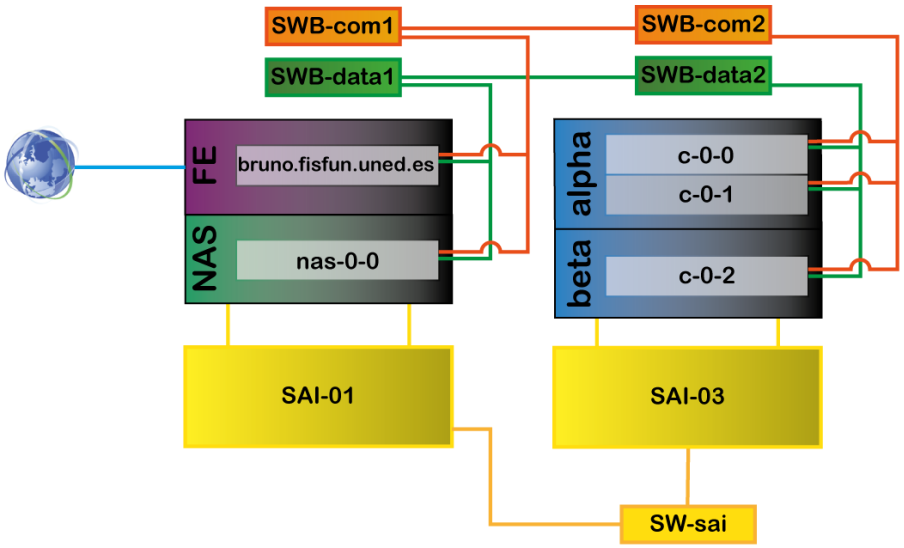
BRUNO es uno de los clústeres de computación que el **CCAR** pone a disposición de sus usuarios. Tiene instalado Rockscluster 7.0, una distribución de GNU/Linux diseñada para entornos HPC basada en CentOS 7.4. **BRUNO** está orientado a estudiantes y labores de iniciación en la adquisición de competencias en HPC. Cuenta con un nodo maestro (*front-end*) que sirve como nodo de login (al que se conectan todos los usuarios) y administra toda la comunicación entre los nodos de cálculo: envía trabajos, gestiona las cuotas de disco y permite instalar fácilmente software, sirviéndolo a todos los equipos mediante instalación en los propios discos o a través de directorios compartidos por NFS.

Toda la red de comunicación entre nodos se controla mediante un par de switches Gigabit Ethernet. Además, los datos de los usuarios (el directorio /home/username) está alojado en un nodo específico con un sistema de almacenamiento en red (NAS), que distribuye la información a los nodos a través de una red ethernet propia y está controlada por otros dos switches Gigabit Ethernet. En este NAS está formado por 5 discos HDD de 4 TB montados en RAID 5 (distribuido con paridad) por software que ofrece un total de 16 TB de capacidad para todos los usuarios.

¡ATENCIÓN!

BRUNO está configurado con un sistema de cuotas de disco para garantizar un uso racional del espacio disponible. Por defecto, los usuarios tienen un «límite blando» de 5 GB en su espacio de usuario, y un «límite duro» de 10 GB. Si un usuario sobrepasa el límite blando, se le enviará una alerta para que corrija la situación en los próximos 30 días. Pasado ese tiempo, o en el momento en el que alcance el límite duro, el usuario no podrá escribir en su espacio de usuario.

El resto de **BRUNO** lo conforman los nodos de cálculo (compute-x-y), organizados en diferentes colas de trabajo, dependiendo de las especificaciones de los nodos. El nombre de estas colas son **ALPHA** y **BETA**. En función del tipo de trabajo a realizar es recomendable utilizar un tipo u otro de nodo.



Estructura de BRUNO, el clúster del CCAR enfocado para estudiantes. El nodo de login (FE) es al que se conectan los usuarios y desde donde se mandan los cálculos científicos. El propio FE envía estos cálculos a los nodos de cálculo (c-x-y). Cada nodo tiene su propio sistema de discos de almacenamiento temporal (scratch). Los datos de usuario (el directorio /home) es común a todos los nodos y está alojado en el NAS.

Cómo trabajar con BRUNO

BRUNO es el clúster de supercomputación del **CCAR** orientado a estudiantes, está pensado para ejecutar trabajos que tengan un alto coste computacional o requieran una paralelización intensiva. Tiene capacidad de cálculo 24/7 y un sistema de almacenamiento redundante tolerante a errores.

En ese sentido, **BRUNO** dispone de tres nodos multinúcleo para procesos en serie y en paralelo. Además, todos los nodos permiten la computación mediante GPGPU, lo que mejora en ciertos casos los tiempos de computación.

Módulos

Dada lo heterogéneo de los cálculos científicos que se realizan, **BRUNO** se organiza en módulos, permitiendo que el usuario active, bajo demanda, aquellos que necesite para computar.

Esencialmente, al activar un módulo cargamos en memoria un conjunto de variables que enlazan a programas o compiladores determinados y a sus librerías correspondientes. Podemos, por ejemplo, cargar el módulo de IntelMPI para que por defecto el compilador sea el propietario de Intel (icc o ifort), las llamadas a mpiexec sean a través del programa de Intel y carguemos en memoria (en la variable **`$LD_LIBRARY_PATH`**) la ruta al conjunto de librerías asociadas a Intel.

Pueden obtenerse todos los módulos disponibles a través del comando **`module avail`**:

código

```
[username@bruno ~]$ module avail
------------------------------ /share/modules --------------------------------
gcc-9.2     gcc-9.3
---------------------- /usr/share/Modules/modulefiles -------------------------
dot             module-info      null           opt-python       rocks-openmpi_ib
module-git      modules          opt-perl       rocks-openmpi    use.own
---------------------------- /etc/modulefiles ------------------------------
mpi/openmpi-x86_64
```

El listado de módulos activos para un usuario en una sesión determinada viene dado por **`module list`**:

código

```
[username@bruno ~]$ module list
Currently Loaded Modulefiles:
  1) rocks-openmpi   2) gcc-9.3
```

Los módulos pueden activarse en el espacio de trabajo de un usuario con **`module load`** y desactivarse con **`module unload`**:

código

```
[username@bruno ~]$ module list
Currently Loaded Modulefiles:
rocks-openmpi
[username@bruno ~]$ module load gcc-9.3
[username@bruno ~]$ module list
Currently Loaded Modulefiles:
  1) rocks-openmpi   2) gcc-9.3
[username@bruno ~]$ module unload rocks-openmpi
[username@bruno ~]$ module list
Currently Loaded Modulefiles:
gcc-9.3
```

Por defecto, **BRUNO** está configurado para tener cargado únicamente el entorno de paralelización OpenMPI (módulo rocks-openmpi) al iniciar sesión.

Colas de trabajo

BRUNO posee un total de 64 hilos de computación accesibles al sistema, repartidos en dos colas de trabajo diferentes:

- **ALPHA**: Dos nodos de computación híbridos en CPU/GPU, con un total de 48 hilos de computación por CPU (24/24), 6 hilos de computación por GPU (3/3) y 32 y 64 GB de RAM respectivamente.
- **BETA**: Un nodo de computación híbrido CPU/GPU, con 16 hilos de computación (CPU), 4096 cores (GPU) y 24 GB de RAM.

Los usuarios, al ser dados de alta, adscriben su pertenencia a un grupo de trabajo específico. Según el grupo de trabajo al que pertenezca el usuario, podrá acceder a un número específico de recursos.

¡ATENCIÓN!

Independientemente del número de hilos accesibles por cada usuario, se limita el número máximo de trabajos ejecutándose en paralelo a un total de 16. Cualquier combinación de trabajos (en serie o en paralelo) que supere el máximo de cuota (por hilos o por trabajos) se pondrá en cola hasta la finalización de los trabajos previos.

Así, si un usuario pertenece a un grupo X, puede mandar, por ejemplo, 4 trabajos con una reserva de 12 hilos por trabajo. Si hay disponibilidad en la cola **ALPHA**, estos trabajos se ejecutarán inmediatamente. Si no hay disponibilidad en esa cola, pero sí en **BETA**, un trabajo se ejecutará en esta última cola mientras los otros tres permanecen a la espera. En caso de no haber disponibilidad en ninguna cola, los trabajos quedarán en espera hasta que alguna de las colas se libere.

Establecer método de trabajo

Dependiendo de si el código a ejecutar está programado en serie o en paralelo el usuario deberá especificar una petición de procesadores u otra. Todos los códigos en serie requieren una petición de un único procesador por trabajo a ejecutar.

Todos los trabajos se envían mediante un sistema de colas (batch) gestionado por el software SLURM. El usuario, tras iniciar sesión en **BRUNO**, puede subir código a su directorio de usuario (a través de scp), editarlo (están instalados los editores vim, vi y nano), compilarlo, etc. Todo ello desde el nodo de login del sistema. Debe evitarse el uso de los nodos de cálculo para este tipo de tareas. Una vez preparado y compilado el código, el procedimiento general de trabajo consiste en enviarle a SLURM (el gestor de recursos) una petición con los recursos solicitados y las instrucciones a ejecutar.

El primer paso es crear un *script* de trabajo (por ejemplo, trabajo.job) especificando las instrucciones para SLURM y los comandos a ejecutar. Las instrucciones específicas de SLURM comienzan con la sentencia (#SBATCH). Los comentarios (ignorados por SLURM) contienen dos almohadillas (##).

código

```
[username@bruno ~]$ nano trabajo.job
#!/bin/bash
##########################################
##       Configuracion del trabajo       ##
##########################################

## COMANDOS OBLIGATORIOS
## Nombre del trabajo
#SBATCH -J prueba
## Nombre del log de salida (%x coge el nombre del trabajo indicado anteriormente)
#SBATCH -o %x.out
## Nombre del log de error (%x coge el nombre del trabajo indicado anteriormente)
#SBATCH -e %x.err
## Número de hilos reservados para el trabajo
#SBATCH --ntasks=1
## Memoria reservada por procesador
#SBATCH --mem-per-cpu=150
## Espacio necesario para almacenar los ficheros temporales (indicado en GB y
entre " ")
DiskSpace="50"

## COMANDOS OPCIONALES
## Especifica a qué cola se mandará el trabajo
## Si se omite se mandará a cualquier cola libre en la que el usuario tenga permiso
##SBATCH --partition ALPHA
## Especifica el nodo al que se manda el trabajo
## Si se omite se mandará al nodo más descargado accesible al usuario
##SBATCH --nodelist=compute-0-0

###############################
##    Comandos a ejecutar    ##
###############################

## NO MODIFICAR esta linea (permite asignar el directorio de scratch)
source /share/bin/jobs/scratch.sh $DiskSpace
```

```
## CARGAR LOS MÓDULOS NECESARIOS (OPCIONAL)
## module unload MODULO_NO_DESEADO
## module load MODULO_DESEADO

## COPIAR FICHEROS AL DIRECTORIO TEMPORAL Y TRASLADARSE A ÉL
cp -r $SLURM_SUBMIT_DIR/* $TMPDIR
cd $TMPDIR
## ----------- EJECUCIÓN DEL PROGRAMA ----------
## Ejemplo 1. Trabajo en serie
## echo "serie" >> ./serie.txt

## Ejemplo 2. Trabajo en paralelo
## ./OMPTest > ./omp.txt

## Ejemplo 3. Trabajo en paralelo
## $SLURM_NTASKS coge automáticamente el numero de hilos reservados anteriormente
## mpirun -np $SLURM_NTASKS ./MPIexample > mpi.txt

## ----------- FIN DE EJECUCIÓN DEL PROGRAMA ----------

## COPIAR LA INFORMACIÓN AL DIRECTORIO DE USUARIO
cp -r * $OUTDIR
```

¡ATENCIÓN!

Para reducir la carga de I/O en la red y mejorar la ejecución del programa, sobre todo en caso de escribir archivos muy grandes (del orden de centenares de MB) es recomendable hacer uso del directorio temporal **`$TMPDIR`** para ejecutar el código. Toda la información de usuario está almacenada en un NAS y es accesible desde bruno a través de NFS. Al lanzar un trabajo, éste se ejecuta en uno de los nodos de cálculon (compute-x-y), haciendo uso de los recursos (procesador, memoria y disco duro) del nodo asignado. En caso de no volcar la información a este nodo de cálculo, todo el proceso de lectura/escritura se realizaría a través de la red del NAS, sobrecargando el conjunto del clúster.

El directorio temporal **$TMPDIR** se crea automáticamente al lanzar el trabajo, dentro de un disco de scratch del nodo de cálculo. Este directorio se elimina automáticamente al finalizar el trabajo, por lo que el usuario debe volcar la información que necesite a su espacio de usuario.

Una vez configurado el *script* trabajo.job, éste debe lanzarse a SLURM mediante el comando **sbatch**:

código

```
[username@bruno ~]$ sbatch trabajo.job
Submitted batch job 76
```

Lo que nos devuelve un identificador del trabajo en ejecución, que se guarda en la variable **$SLURM_JOB_ID** (en el ejemplo, 76).

Si por algún motivo, se quiere cancelar la ejecución del trabajo, se usa el comando **scancel**, indicando el identificador del trabajo (en este caso, 76).

código

```
[username@bruno ~]$ scancel 76
```

Comprobar estado de un trabajo

Una vez mandado un trabajo a la cola de envíos, este queda en estado PD (*pending*) hasta que el asignador de trabajos encuentra disponibilidad para lanzarlo. En ese momento pasa a un estado T (*transferring*), hasta que el nodo de computación toma el control del trabajo y lo ejecuta, entrando en estado R (*running*). Los distintos estados pueden comprobarse ejecutando el comando **squeue**:

código

```
[username@bruno ~]$ squeue
JOBID   PARTITION   NAME      USER       ST      TIME     NODES    NODELIST(REASON)
75      ALPHA       prueba    username   R       0:02     1        compute-0-2
```

`squeue` muestra, de izquierda a derecha:

- JOBID: El número identificador del trabajo.
- PARTITION: La cola donde está corriendo el trabajo.
- NAME: El nombre del trabajo.
- USER: El usuario que ha mandado el trabajo.
- ST: El estado del trabajo.
- TIME: El tiempo que el trabajo lleva ejecutándose.
- NODE: El número de nodos en los que está corriendo el trabajo.
- NODELIST (REASON): El nombre de los nodos en los que está corriendo, o las razones por las que no lo está haciendo.

¡ATENCIÓN!

Si por algún motivo el trabajo no pudiera ejecutarse, quedaría en estado PD indefinidamente, indicando en la sección NODELIST (REASON) la razón. Si la razón fuera, por ejemplo, que el usuario no tiene permisos para lanzar trabajos en la cola que ha especificado, el trabajo quedaría en espera indefinidamente. En estos casos lo mejor es cancel el trabajo con `scancel`.

Para una visión pormenorizada del estado de BRUNO puede usarse el comando `sstatus`, el cual muestra información de cada uno de los nodos de cada una de las colas.

código

```
[username@bruno ~]$ sstatus

---------------------------------------------------------------
PARTITION     NODELIST              CPUS(A/I/O/T)
ALPHA            compute-0-0         0/24/0/24
```

```
---------------------------------------
JOBID  NAME        USER      ST         TIME    NODES    CPUS     NODELIST(REASON)
-------------------------------------------------------------------
PARTITION     NODELIST              CPUS(A/I/O/T)
ALPHA            compute-0-1         6/18/0/24
---------------------------------------
JOBID  NAME        USER      ST         TIME    NODES    CPUS     NODELIST(REASON)
78     prueba      username  R           0:02    1       6        compute-0-1
-------------------------------------------------------------------
PARTITION     NODELIST           CPUS(A/I/O/T)
ALPHA            compute-0-2         0/16/0/16
---------------------------------------
JOBID  PARTITION   NAME      USER       ST     TIME     NODES    NODELIST(REASON)
```

Como se puede ver en el código, el trabajo prueba está corriendo en la cola **ALPHA**, en el nodo compute-0-1, habiendo reservado 6 hilos para su ejecución.

Resumen de comandos

Comandos de módulos

`module avail`	Ver los módulos disponibles
`module list`	Ver los módulos cargados actualmente
`module load + módulo`	Cargar un módulo
`module unload + módulo`	Descargar un módulo

Comandos de trabajos

`sbatch + nombre_trabajo`	Lanzar un trabajo
`scancel + ID_trabajo`	Cancelar un trabajo
`squeue`	Ver el estado de un trabajo
`sstatus`	Ver información de las colas

Variables de entorno

`$HOME`	El directorio que contiene los datos del usuario (por ejemplo, /home/username). Está alojado físicamente en el NAS, y es accesible desde cualquier nodo de login o de cálculo.
`$SLURM_SUBMIT_DIR`	El directorio desde el que se lanza el trabajo (por ejemplo, /home/username/MyJob).
`$SLURM_JOB_ID`	El identificador del trabajo (por ejemplo, 37).
`$TMPDIR`	El directorio temporal de trabajo (por ejemplo, /state/partition1/username/MyJob). Este directorio se crea al lanzar el trabajo en el nodo de cálculo, y es donde se almacena toda la información temporal. El directorio se elimina automáticamente al finalizar el trabajo.