Centro de Computación de Alto Rendimiento

CCAR_UNED

ALICE – Manual de usuario

Versión 1.1

#SOMOS2030

https://ccar.uned.es

CLÚSTER ALICE – Manual de Usuario (v.1.1)

Centro de Computación de Alto Rendimiento CCAR

Junio 2022

Sumario

Introducción

El Centro de Computación de Alto Rendimiento de la Facultad 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* ofrecerá servicios de asesoría, adquisición de equipos e infraestructuras, soporte y mantenimiento de hardware y software de altas prestaciones.

Estructura

ALICE, el clúster del CCAR orientado a investigadores, cuenta con un nodo maestro que administra toda la comunicación entre los nodos de cálculo y almacenamiento: envía trabajos, gestiona las cuotas de disco de los usuarios y permite instalar fácilmente software, sirviéndolo a los nodos de cálculo mediante instalación en los propios discos o a través de directorios compartidos por NFS.

El sistema operativo instalado en *ALICE* es Rockscluster 6.2, una distribución de GNU/Linux diseñada para entornos HPC basada en CentOS 6.8.

Toda la red de comunicación entre nodos se controla mediante un par de switches Gigabit Ethernet. El espacio de almacenamiento de usuario (directorio /home), así como gran parte del software instalado, se comparte a través de nodos de almacenamiento en red (NAS) que distribuyen la información a través de una red propia y está controlada por otro par de switches Gigabit Ethernet. Cada NAS cuenta con un sistema de discos configurados en RAID5 (distribuidos por paridad) con protección frente a errores de disco. El NAS principal tiene una capacidad total de 22 TB para todos los usuarios. Además, un segundo NAS con 16 TB se utiliza para almacenamiento a largo plazo.

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

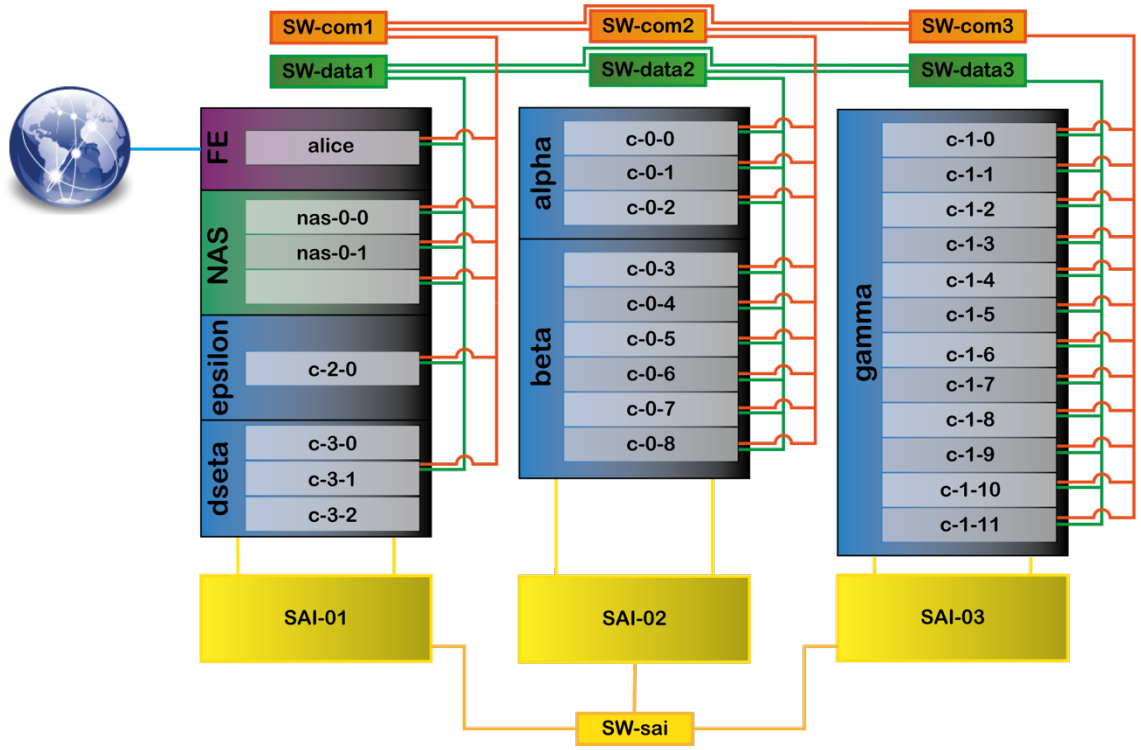

Figura 1. Esquema de ALICE, incluyendo topología de red, nodos y sistemas de alimentación ininterrumpida.

Cómo trabajar con ALICE

ALICE es el clúster de supercomputación del **CCAR** orientado a investigadores. Está pensado para ejecutar trabajos que tengan un alto coste computacional o requieran una paralelización intensiva.

En ese sentido, *ALICE* dispone de 3 nodos mononúcleo de alta frecuencia (cola **alpha**) para procesos en serie y 22 nodos multinúcleo para procesos en paralelo (colas **beta**, **gamma**, **epsilon** y **dseta**). Además, las colas **alpha**, **beta** y **dseta**, con 12 nodos en su conjunto, permiten la computación mediante GPGPU, lo que mejora en ciertos casos los tiempos de computación.

Módulos

Dada la diferente casuística existente, *ALICE* se organiza en módulos de trabajo, permitiendo que el usuario active, bajo demanda, aquellos módulos que necesite para realizar cálculos.

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 de entorno `$LD_LIBRARY_PATH`) el conjunto de librerías asociadas a Intel.

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

código

```
[username@alice ~]$ 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@alice ~]$ 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@alice ~]$ module list
Currently Loaded Modulefiles:
rocks-openmpi

[username@alice ~]$ module load gcc-9.3

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

[username@alice ~]$ module unload rocks-openmpi

[username@alice ~]$ module list
Currently Loaded Modulefiles:
gcc-9.3
```

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

Colas de trabajo

ALICE posee más de 1000 hilos de computación accesibles al sistema, repartidos en cinco colas de trabajo diferentes:

- **alpha:** Tres nodos de computación híbridos en CPU/GPU, con un total de 36 hilos de computación por CPU y 64 GB de RAM por nodo. Esta cola cuenta con tres GTX 1080 Ti y cuatro RTX 2080 Ti como GPUs.
- **beta:** Seis nodos de computación híbridos CPU/GPU, con 288 hilos de computación por CPU y entr 64 GN y 128 GB de RAM por nodo. Esta cola cuenta con nueve GTX 1080 Ti y seis RTX 3080 como GPUs.
- **gamma:** Doce nodos multinúcleos de computación por CPU, con un total de 480 hilos de computación por CPU y entre 64 GB y 96 GB de RAM por nodo.
- **epsilon:** Un nodo multinúcleo con 64 hilos de computación por CPU y 192 GB de RAM.
- **dseta:** Tres nodos de computación híbrida CPU/GPU, con un total de 176 hilos de computación por CPU y entre 128 GB y 192 GB de RAM. Esta cola cuenta además con dos GTX 1080 Ti y 2 A4000 como GPUs.

Toda la información actualizada sobre las características de cada nodo puede encontrarse en la página web del *CCAR* **https://ccar.uned.es/hpc**.

Los usuarios, al ser dados de alta, adscriben su pertenencia a un grupo de trabajo concreto. Según el grupo de trabajo al que pertenezca el usuario, podrá acceder a un número específico de recursos. En general, todas las colas de trabajo están disponibles (con distinta prioridad) a todos los usuarios. Es recomendable utilizar las colas de trabajo asignadas al grupo, ya que en estos nodos la prioridad en el envío de los trabajos es máxima. El acceso a los recursos de otras colas está limitado al uso de unos pocos hilos de computación

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 un gestor de recursos: *Grid Engine (SGE)*. El usuario, tras iniciar sesión en *ALICE*, 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 maestro 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 SGE una petición con los recursos solicitados y las instrucciones a ejecutar.

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

código

```
[username@alice ~]$ cat trabajo.job
#!/bin/bash

##################################
##  Configuracion del trabajo  ##
##################################

## Utilizar la Shell de bash (recomendable)
#$ -S /bin/bash

## Importar las variables de entorno del sistema
## como $PWD, $USERNAME, $PATH, etc. (muy recomendable)
#$ -V

## Ejecutar el trabajo en el directorio actual (recomendable)
#$ -cwd
```

```
## Ejecutar en una cola determinada (recomendable)
#$ -q NOMBRE_COLA

## Activar paralelización (obligatorio para trabajos en paralelo
## PEV   = orte, mpi, mpich
## NPROC = número de procesadores
#$ -pe PEV NPROC

## Nombre del trabajo (opcional)
#$ -N NOMBRE_DEL_TRABAJO

## Log de salida (opcional)
#$ -o NOMBRE_DEL_ARCHIVO.log

## Log de error (opcional)
#$ -e NOMBRE_DEL_ARCHIVO.log

## Fusionar salida y error en el mismo archivo (opcional)
#$ -j y

## Reserva de espacio en disco (en GB)
DiskSpace="GB_RESERVADOS"

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

## Asignación de directorio de trabajo (no modificar)
Source /usr/local/jobs/scratch.sh $DiskSpace

## Cargar los módulos necesarios (opcional)
## module unload MODULO_NO_DESEADO
## module load MODULO_DESEADO

## Volcar toda la información a un directorio temporal
cp -r $SGE_O_WORKDIR/* $TMPDIR
cd $TMPDIR

## Trabajo a ejecutar
## Descomentar lo que proceda
## A. Trabajo en paralelo
##    A.1. MPI
##         mpirun -np NPROC ./ejecutable
##    A.2. OpenMP
##         ./ejecutable
## B. Trabajo en serie
##    ./ejecutable

## Volcar toda la información al home del usuario
OUTDIR="$SGE_O_WORKDIR/$JOB_ID"
mkdir -p $OUTDIR
cp -r * $OUTDIR
```

La línea 14 especifica la cola de envío (**alpha**, **beta**, **gamma**, **epsilon** o **dseta**). En caso de omisión el trabajo irá a cualquier cola disponible. Es posible especificar un nodo de cálculo concreto utilizando la sintaxis `COLA@NODO`. Por ejemplo:

```
#$ -q gamma@compute-0-9
```

Mandará el trabajo al nodo `compute-0-9` de la cola **gamma**.

La línea 18 especifica, para trabajos en paralelo, un entorno de paralelización (orte, mpi o mpich, en función del código a ejecutar) y el número de hilos reservados. Por ejemplo:

```
#$ -pe mpi 16
```

Reservaría 16 hilos de computación para ejecutar con mpi.

La línea 28 permite especificar las necesidades de disco. Cada nodo de trabajo dispone de dos discos para espacio de almacenamiento temporal. Un disco de estado sólido de lectura y escritura rápida, y un disco mecánico de gran capacidad. En función del espacio requerido, el trabajo se ejecutará en un disco u otro para optimizar los tiempos de cálculo y garantizar el cumplimiento de los requisitos de disco.

Las líneas 29 a 52 son la ejecución propia del trabajo. Cada vez que se inicia un trabajo en un nodo de cálculo, se crea en ese nodo un directorio temporal en /state/partition1 (disco mecánico) o /state/partition2 (disco de estado sólido) en función de los requisitos de espacio, con un identificador dado por la variable `$JOBID.QUEUE`. Este directorio es accesible por el usuario que ha lanzado el trabajo, y se elimina cuando acaba la ejecución. Puede referenciarse a él dentro del script sin más que llamar a la variable `$TMPDIR`.

Para reducir la carga de entrada y salida en la red y mejorar la ejecución del programa, sobre todo en caso de escribir archivos muy grandes (del orden de GB) es recomendable hacer uso del directorio temporal `$TMPDIR` para ejecutar el trabajo. Toda la información de usuario está almacenada en un NAS y es accesible a todos los nodos a través de NFS. Al lanzar un trabajo, este se ejecuta en uno de los nodos de cálculo (`compute-x-y`) haciendo uso de los recursos (procesador, memoria y disco duro) del nodo asignado. En caso de no volcar la información a este nodo, todo el proceso de lectura/escritura se haría a través de la red del NAS, sobrecargando el clúster. En general, deben volcarse al nodo de cálculo únicamente los archivos necesarios para la ejecución del trabajo. Para ello es recomendable que cada trabajo esté en un directorio específico, que viene identificado en la línea 38 por la variable `$SGE_O_WORKDIR`.

El bloque 40-48 contiene diversos ejemplos para ejecutar el código compilado `./ejecutable`. El primer de ellos utiliza las librerías MPI, especificando el número de procesadores reservados por SGE. Es importante que la variable `NPROC` (línea 44) se sustituya por la reserva realizada en el entorno de paralelización (línea 18). Por ejemplo:

```
mpirun -np 16 ./ejecutable
```

El segundo ejemplo utiliza las librerías de OpenMP (para las que no es necesario especificar el número de procesadores). El tercero muestra un ejemplo para trabajo en serie, en cuyo caso basta utilizar directamente el ejecutable.

Envío, estado y cancelación de trabajos

Una vez configurado el script `trabajo.job`, este debe lanzarse a SGE mediante la instrucción *qsub*:

código

```
1  [username@alice ~]$ qsub trabajo.job
2  Your job 121 ("trabajo.job") has been submitted
```

Lo que nos devuelve un identificador del trabajo en ejecución, 121.

Una vez mandado un trabajo a la cola de envíos, este queda en espera (estado `w`, waiting) hasta que el asignador de recursos encuentra disponibilidad para lanzarlo. En ese momento pasa a un estado `t` (transfering) hasta que el nodo de cálculo toma el control del trabajo y lo ejecuta `r` (running). Los distintos estados pueden comprobarse con el comando *qstat*:

código

```
1 [username@alice ~]$ qstat
2 job-ID prior name   user      state submit/start at queue slots ja-task-ID
3 -----------------------------------------------------------------------------
4 121 0.55500 CG      username r 01/18/2017 18:57:25 alpha@compute-0-2.local    8
5 124 0.55500 mpi-r.q username r 01/18/2017 18:59:10 beta@compute-0-3.local     8
6 125 0.55500 mpi-r.q username r 01/18/2017 18:59:10 beta@compute-0-3.local     8
7 126 0.55500 mpi-r.q username r 01/18/2017 18:59:10 gamma@compute-0-10.local   8
```

qstat muestra, de izquierda a derecha:

- El identificador del trabajo (job-ID).

- La prioridad de ejecución (prior), factor dependiente de la cola a la que se ha enviado, el número de trabajos enviados por el usuarios, su cuota asignada, y los recursos solicitados.
- El nombre del trabajo (name), tomado a través de la variable -N del script.
- El usuario que ha lanzado el trabajo (user)
- El estado del trabajo (state)
- El momento de envío o inicio (submit/start at)
- La cola de trabajo y el nodo asignado (queue)
- El número de hilos reservados (slots).

Un trabajo ya enviado puede cancelarse mediante *qdel*. Por ejemplo:

código

```
1  [username@alice ~]$ qdel 162
```

Detendría la ejecución del trabajo 162.

Un estado `E` es un estado erróneo. Puede verse más información con *qstat*:

código

```
1  [username@alice ~]$ qstat -explain E
```

Un estado `a` es un estado de alarma. Puede verse más información utilizando el mismo comando:

código

```
1  [username@alice ~]$ qstat -explain a
```

Para una vision pormenorizada del estado de *ALICE* puede usarse el comando *qstat -f*, el cual muestra información de cada uno de los nodos de cada una de las colas:

código

```
1  [username@alice ~]$ qstat -f
2  queuename                      qtype resv/used/tot. load_avg arch      states
```

```
3  ---------------------------------------------------------------------------
4  alpha@compute-0-1.local        BIP   0/0/24         0.08     linux-x64     E
5  ---------------------------------------------------------------------------
6  alpha@compute-0-2.local        BIP   0/8/24         -NA-     linux-x64     au
7  121 0.55500 CG-Walls-0 jatorre      dr    01/18/2017 18:57:25     8
8  ---------------------------------------------------------------------------
9  beta@compute-0-3.local         BIP   0/12/12        0.08     linux-x64
10 133 0.55500 mpi-ring.q jatorre      r     01/18/2017 19:53:40     8
11 134 0.55500 mpi-ring.q jatorre      r     01/18/2017 19:53:40     4
12 ---------------------------------------------------------------------------
13 gamma@compute-0-10.local       BIP   0/20/40        6.04     linux-x64
14 134 0.55500 mpi-ring.q jatorre      r     01/18/2017 19:53:40     4
15 135 0.55500 mpi-ring.q jatorre      r     01/18/2017 19:53:40     8
16 136 0.55500 mpi-ring.q jatorre      r     01/18/2017 19:53:40     8
17 ---------------------------------------------------------------------------
18 gamma@compute-0-11.local       BIP   0/0/40         6.06     linux-x64
19 ---------------------------------------------------------------------------
20 gamma@compute-0-12.local       BIP   0/0/40         0.13     linux-x64     E
21 ---------------------------------------------------------------------------
22 gamma@compute-0-9.local        BIP   0/0/40         16.04    linux-x64
```

donde se muestra, en este ejemplo, que

- `compute-0-1`, de la cola **alpha**, está en un estado de error `E` con 0 hilos reservados y 0 en uso, del total de 24. El nodo tiene una carga media de 0.08.
- `compute-0-2`, de la cola **alpha**, está en un estado desconocido (`u`) de alarma (`a`), con 8 hilos en uso de los 24 totales y un trabajo (121) ejecutado por el usuario `jatorre` que está corriendo (`r`) pero muerto (`d`), con 8 hilos reservados.
- `compute-0-3`, de la cola **beta** está completamente en uso (12 hilos de 12 disponibles) pero sin carga (0.08). Los dos trabajos (133 y 134) está corriendo con 8 y 4 hilos, respectivamente.
- …

Con la opción `-u '*'` podemos ver la información extendida a todos los usuarios de *ALICE*:

código

```
1  [username@alice ~]$ qstat -f -u '*'
```

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

`qsub + nombre_trabajo`	Lanzar un trabajo
`qdel + ID_trabajo`	Cancelar un trabajo
`qstat`	Ver el estado de un trabajo
`qstat -f`	Ver información de las colas