

Cluster de Computación de Altas Prestaciones (HPC) ctcomp3

[Video de la presentación del servicio \(7/3/22\)](#)

Descripción

El clúster está compuesto en la parte de cómputo por:

- 9 servidores para cómputo general.
- 1 “fat node” para trabajos que requieran mucha memoria.
- 4 servidores para computo con GPU.

Los usuarios solo tienen acceso directo al nodo de login, de prestaciones más limitadas y que no debe usarse para computar.

Todos los nodos están interconectados por una red a 10Gb.

Hay un almacenamiento distribuido accesible desde todos los nodos con 220 TB de capacidad conectado mediante una doble red de fibra de 25Gb.

Nombre	Modelo	Procesador	Memoria	GPU
hpc-login2	Dell R440	1 x Intel Xeon Silver 4208 CPU @ 2.10GHz (8c)	16 GB	-
hpc-node[1-2]	Dell R740	2 x Intel Xeon Gold 5220 @2,2 GHz (18c)	192 GB	-
hpc-node[3-9]	Dell R740	2 x Intel Xeon Gold 5220R @2,2 GHz (24c)	192 GB	-
hpc-fat1	Dell R840	4 x Xeon Gold 6248 @ 2.50GHz (20c)	1 TB	-
hpc-gpu[1-2]*	Dell R740	2 x Intel Xeon Gold 5220 CPU @ 2.20GHz (18c)	192 GB	2x Nvidia Tesla V100S
hpc-gpu3	Dell R7525	2 x AMD EPYC 7543 @2,80 GHz (32c)	256 GB	2x Nvidia Ampere A100 40GB
hpc-gpu4	Dell R7525	2 x AMD EPYC 7543 @2,80 GHz (32c)	256 GB	1x Nvidia Ampere A100 80GB

* Son ctgpgpu7 y 8. Se integrarán próximamente en cluster.

Conexión al sistema

Para acceder al clúster, hay que solicitarlo previamente a través de [formulario de incidencias](#). Los usuarios que no tengan permiso de acceso recibirán un mensaje de “contraseña incorrecta”.

El acceso se realiza mediante una conexión SSH al nodo de login:

```
ssh <nombre_de_usuario>@hpc-login2.inv.usc.es
```

Almacenamiento, directorios y sistemas de ficheros

No se hace copia de seguridad de ninguno de los sistemas de ficheros del cluster!!

El HOME de los usuarios en el cluster está en el sistema compartido de ficheros, por lo que es accesible desde todos los nodos del cluster. Ruta definida en la variable de entorno \$HOME. Cada nodo tiene una partición local de 1 TB para scratch, que se borra al terminar cada trabajo. Se puede acceder mediante la variable de entorno \$LOCAL_SCRATCH en los scripts. Para datos que deban ser compartidos por grupos de usuarios, hay que solicitar la creación de una carpeta en el almacenamiento compartido que solo será accesible por los miembros del grupo.

Directorio	Variable	Punto de montaje	Capacidad
Home	\$HOME	/mnt/beegfs/home/<username>	220 TB*
Scratch local	\$LOCAL_SCRATCH	varía	1 TB
Carpeta de grupo	\$GRUPOS/<nombre>	/mnt/beegfs/groups/<nombre>	220 TB*

* el almacenamiento es compartido

AVISO IMPORTANTE

El sistema compartido de archivos tiene un mal rendimiento cuando trabaja con muchos archivos de tamaño pequeño. Para mejorar el rendimiento en ese tipo de escenarios hay que crear un sistema de archivos en un fichero de imagen y montarlo para trabajar directamente sobre él. El procedimiento es el siguiente:

- Crear el fichero de imagen en tu home:

```
## truncate image.name -s SIZE_IN_BYTES
truncate ejemplo.ext4 -s 20G
```

- Crear un sistema de archivos en el fichero de imagen:

```
## mkfs.ext4 -T small -m 0 image.name
## -T small opciones optimizadas para archivos pequeños
## -m 0 No reservar espacio para root
mkfs.ext4 -T small -m 0 ejemplo.ext4
```

- Montar la imagen (usando SUDO) con el script `mount_image.py` :

```
## Por defecto queda montada en /mnt/imagenes/<username>/ en modo solo lectura.
sudo mount_image.py ejemplo.ext4
```

- Para desmontar la imagen usar el script `umount_image.py` (usando SUDO)

El script de montaje tiene estas opciones:

```
--mount-point path <-- (opcional) Con esta opción crea subdirectorios por debajo de /mnt/imagenes/<username>/<path>
--rw <-- (opcional) Por defecto se monta readonly, con esta
```

```
opción se monta readwrite.
```

El script de desmontaje tiene estas opciones:

```
solo admite como parámetro opcional el mismo path que hayas usado para el montaje con la opción  
--mount-point <-- (opcional)
```

Transferencia de ficheros y datos

SCP

Desde tu máquina local al cluster:

```
scp filename <username>@hpc-login2:/<ruta>
```

Desde el cluster a tu máquina local:

```
scp filename <username>@<hostname>:/<ruta>
```

[Página del manual de SCP](#)

SFTP

Para transferir múltiples archivos o para navegar por el sistema de archivos.

```
<hostname>:~$ sftp <user_name>@hpc-login2  
sftp>  
sftp> ls  
sftp> cd <path>  
sftp> put <file>  
sftp> get <file>  
sftp> quit
```

[Página del manual de SFTP](#)

RSYNC

[Documentación de RSYNC](#)

SSHFS

Requiere la instalación del paquete sshfs.

Permite por ejemplo montar el home del equipo del usuario en hpc-login2:

```
## Montar
sshfs <username>@ctdeskxxx.inv.usc.es:/home/<username> <punto_de_montaje>
## Desmontar
fusermount -u <punto_de_montaje>
```

[Página del manual de SSHFS](#)

Software disponible

Todos los nodos tienen el software básico que se instala por defecto con AlmaLinux 8.4, particularmente:

- GCC 8.5.0
- Python 3.6.8
- Perl 5.26.3

Para usar cualquier otro software no instalado en el sistema u otra versión del mismo hay tres opciones:

1. Usar Modules con los módulos que ya están instalados (o solicitar la instalación de un nuevo módulo si no está disponible)
2. Usar un contenedor (uDocker o Apptainer/Singularity)
3. Usar Conda

Un módulo es la solución más sencilla para usar software sin modificaciones o dependencias difíciles de satisfacer.

Un contenedor es ideal cuando las dependencias son complicadas y/o el software está muy personalizado. También es la mejor solución si lo que se busca es reproducibilidad, facilidad para su distribución y trabajo en equipo.

Conda es la mejor solución si lo que se necesita es la última versión de una librería o programa o paquetes no disponibles de otra forma.

Uso de modules/Lmod

[Documentación de Lmod](#)

```
# Ver los módulos disponibles:
module avail
# Cargar un módulo:
module <nombre_modulo>
# Descargar un módulo:
module unload <nombre_modulo>
# Ver módulos cargados en tu entorno:
module list
# Puede usarse ml como abreviatura del comando module:
ml avail
```

```
# Para obtener informaci3n sobre un m3dulo:  
ml spider <nombre_modulo>
```

Ejecuci3n de contenedores de software

uDocker

[Manual de uDocker](#)

uDocker est3 instalado como un m3dulo, as3 que es necesario cargarlo en el entorno:

```
ml uDocker
```

Apptainer/Singularity

[Documentaci3n de Apptainer/Singularity](#)

Apptainer/Singularity est3 instalado en el sistema de cada nodo, por lo que no es necesario hacer nada para usarlo.

CONDA

[Documentaci3n de Conda](#)

Miniconda es la versi3n m3nima de Anaconda y solo incluye el gestor de entornos conda, Python y unos pocos paquetes necesarios. A partir de ah3 cada usuario solo descarga e instala los paquetes que necesita.

```
# Obtener miniconda  
wget https://repo.anaconda.com/miniconda/Miniconda3-py39_4.11.0-Linux-x86_64.sh  
# Instalarlo  
sh Miniconda3-py39_4.11.0-Linux-x86_64.sh
```

Uso de SLURM

El gestor de colas en el cluster es [SLURM](#) .

El t3rmino CPU identifica a un core f3sico de un socket. El hyperthreading est3 desactivado, por lo que cada nodo tiene disponibles tantas CPU como (n3 sockets) * (n3 cores f3sico por socket) tenga.

Recursos disponibles

```
hpc-login2 ~]$ sinfo -e -o "%30N %20c %20m %20f %30G " --sort=N  
# Hay un alias para este comando:  
hpc-login2 ~]$ ver_recursos  
NODELIST                                CPUS                                MEMORY
```

AVAIL_FEATURES	GRES		
hpc-fat1		80	1027273
cpu_intel	(null)		
hpc-gpu[1-2]		36	187911
cpu_intel	gpu:V100S:2		
hpc-gpu3		64	253282
cpu_amd	gpu:A100_40:2		
hpc-gpu4		64	253282
cpu_amd	gpu:A100_80:1(S:0)		
hpc-node[1-2]		36	187645
cpu_intel	(null)		
hpc-node[3-9]		48	187645
cpu_intel	(null)		

Nodos

Un nodo es la unidad de computación de SLURM, y se corresponde con un servidor físico.

```
# Mostrar la información de un nodo:
hpc-login2 ~]$ scontrol show node hpc-node1
NodeName=hpc-node1 Arch=x86_64 CoresPerSocket=18
  CPUAlloc=0 CPUTot=36 CPULoad=0.00
  AvailableFeatures=cpu_intel
  ActiveFeatures=cpu_intel
  Gres=(null)
NodeAddr=hpc-node1 NodeHostName=hpc-node1 Version=21.08.6
OS=Linux 4.18.0-305.el8.x86_64 #1 SMP Wed May 19 18:55:28 EDT 2021
RealMemory=187645 AllocMem=0 FreeMem=166801 Sockets=2 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=defaultPartition
BootTime=2022-03-01T13:13:56 SlurmdStartTime=2022-03-01T15:36:48
LastBusyTime=2022-03-07T14:34:12
CfgTRES=cpu=36,mem=187645M,billing=36
AllocTRES=
CapWatts=n/a
CurrentWatts=0 AveWatts=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Particiones

Las particiones en SLURM son grupos lógicos de nodos. En el cluster hay una única partición a la que pertenecen todos los nodos, por lo que no es necesario especificarla a la hora de enviar trabajos.

```
# Mostrar la información de las particiones:
hpc-login2 ~]$ sinfo
defaultPartition* up infinite 11 idle hpc-fat1,hpc-gpu[3-4],hpc-
node[1-9]
# Cuando se incorporen al cluster ctgpgpu7 y 8 apareceran como los nodos
```

hpc-gpu1 y 2 respectivamente.

Trabajos

Los trabajos en SLURM son asignaciones de recursos a un usuario durante un tiempo determinado. Los trabajos se identifican por un n3mero correlativo o JOBID. Un trabajo (JOB) consiste en uno o m3s pasos (STEPS), cada uno consistente en una o m3s tareas (TASKS) que usan una o m3s CPU. Hay un STEP por cada programa que se ejecute de forma secuencial en un JOB y hay un TASK por cada programa que se ejecute en paralelo. Por lo tanto en el caso m3s simple como por ejemplo lanzar un trabajo consistente en ejecutar el comando hostname el JOB tiene un 3nico STEP y una 3nica TASK.

Sistema de colas (QOS)

La cola a la que se env3e cada trabajo define la prioridad, los l3mites y tambi3n el "coste" relativo para el usuario.

```
# Mostrar las colas
hpc-login2 ~]$ sacctmgr show qos
# Hay un alias que muestra solo la informaci3n m3s relevante:
hpc-login2 ~]$ ver_colas
```

Name	Priority	Flags	UsageFactor
MaxTRES	MaxWall	MaxTRESPU	MaxJobsPU
			MaxSubmitPU
regular	100	DenyOnLimit	1.000000
cpu=200,gres/gpu=1,node=4	4-04:00:00		10 50
interactive	200	DenyOnLimit	1.000000
node=1 04:00:00	node=1		1 1
urgent	300	DenyOnLimit	2.000000
gres/gpu=1,node=1	04:00:00	cpu=36	5 15
long	100	DenyOnLimit	1.000000
gres/gpu=1,node=4	8-08:00:00		
large	100	DenyOnLimit	1.000000
cpu=200,gres/gpu=2	4-04:00:00		10 25
admin	500		0.000000

Priority: es la prioridad relativa de cada cola.

DenyOnLimit: el trabajo no se ejecuta si no cumple los l3mites de la cola

UsageFactor: el coste relativo para el usuario de ejecutar un trabajo en esa cola

MaxTRES: l3mites por cada trabajo

MaxWall: tiempo m3ximo que puede estar el trabajo en ejecuci3n

MaxTRESPU: l3mites globales por usuario

MaxJobsPU: N3mero m3ximo de trabajos que un usuario puede tener en ejecuci3n.

MaxSubmitPU: N3mero m3ximo de trabajos que un usuario puede tener en total encolados y en ejecuci3n.

Envío de un trabajo al sistema de colas

Especificación de recursos

Por defecto, si se envía un trabajo sin especificar nada el sistema lo envía a la QOS por defecto (regular) y le asigna un nodo, una CPU y toda la memoria disponible. El límite de tiempo para la ejecución del trabajo es el de la cola (4 días y 4 horas). Esto es muy ineficiente, lo ideal es especificar en la medida de lo posible al menos tres parámetros a la hora de enviar los trabajos:

1. El número de nodos (-N o --nodes), tareas (-n o --ntasks) y/o CPU por tarea (-c o --cpus-per-task).
2. La memoria (--mem) por nodo o la memoria por cpu (--mem-per-cpu).
3. El tiempo estimado de ejecución del trabajo (--time)

A mayores puede ser interesante añadir los siguientes parámetros:

-J	--job-name	Nombre para el trabajo. Por defecto: nombre del ejecutable
-q	--qos	Nombre de la cola a la que se envía el trabajo. Por defecto: regular
-o	--output	Fichero o patrón de fichero al que se redirige toda la salida estandar y de error.
	--gres	Tipo y/o número de GPUs que se solicitan para el trabajo.
-C	--constraint	Para especificar que se quieren nodos con procesadores Intel o AMD (cpu_intel o cpu_amd)
	--exclusive	Para solicitar que el trabajo no comparta nodos con otros trabajos.
-w	--odelist	Lista de nodos en los que ejecutar el trabajo

Cómo se asignan los recursos

Por defecto el método de asignación entre nodos es la asignación en bloque (se asignan todos los cores disponibles en un nodo antes de usar otro). El método de asignación por defecto dentro de cada nodo es la asignación cíclica (se van repartiendo por igual los cores requeridos entre los sockests disponibles en el nodo).

Calculo de la prioridad

Cuando se envía un trabajo al sistema de colas, lo primero que ocurre es que se comprueba si los recursos solicitados entran dentro de los límites fijados en la cola correspondiente. Si supera alguno se cancela el envío.

Si hay recursos disponibles el trabajo se ejecuta directamente, pero si no es así se encola. Cada trabajo tiene asignada una prioridad que determina el orden en que se ejecutan los trabajos de la cola cuando quedan recursos disponibles. Para determinar la prioridad de cada trabajo se ponderan 3 factores: el tiempo que lleva esperando en la cola (25%), la prioridad fija que tiene la cola(25%) y el fairshare del usuario (50%).

El fairshare es un cálculo dinámico que hace SLURM para cada usuario y es la diferencia entre los recursos asignados y los recursos consumidos a lo largo de los últimos 14 días.

```
hpc-login2 ~]$ sshare -l
  User  RawShares  NormShares  RawUsage  NormUsage  FairShare
```



```

-----
                1.000000    2872400    0.500000
                1    0.500000    2872400    1.000000    0.250000
user_name      100    0.071429    4833    0.001726    0.246436

```

RawShares: es la cantidad de recursos en t3rminos absolutos asignada al usuario. Es igual para todos los usuarios.

NormShares: Es la cantidad anterior normalizada a los recursos asignados en total.

RawUsage: Es la cantidad de segundos/cpu consumida por todos los trabajos del usuario.

NormUsage: Cantidad anterior normalizada al total de segundos/cpu consumidos en el cluster.

FairShare: El factor FairShare entre 0 y 1. Cuanto mayor uso del cluster, m3s se aproximará a 0 y menor ser3 la prioridad.

Envío de trabajos

1. salloc
2. srun
3. sbatch

1. SALLOC

Sirve para obtener de forma inmediata una asignaci3n de recursos (nodos). En cuanto se obtiene se ejecuta el comando especificado o una shell en su defecto.

```

# Obtener 5 nodos y lanzar un trabajo.
hpc-login2 ~]$ salloc -N5 myprogram
# Obtener acceso interactivo a un nodo (Pulsar Ctrl+D para terminar el
acceso):
hpc-login2 ~]$ salloc -N1

```

2. SRUN

Sirve para lanzar un trabajo paralelo (es preferible a usar mpirun). Es interactivo y bloqueante.

```

# Lanzar un hostname en 2 nodos
hpc-login2 ~]$ srun -N2 hostname
hpc-node1
hpc-node2

```

3. SBATCH

Sirve para enviar un script al sistema de colas. Es de procesamiento por lotes y no bloqueante.

```

# Crear el script:
hpc-login2 ~]$ vim trabajo_ejemplo.sh
#!/bin/bash
#SBATCH --job-name=prueba           # Job name
#SBATCH --nodes=1                   # -N Run all processes on a single
node
#SBATCH --ntasks=1                 # -n Run a single task
#SBATCH --cpus-per-task=1          # -c Run 1 processor per task
#SBATCH --mem=1gb                  # Job memory request
#SBATCH --time=00:05:00            # Time limit hrs:min:sec

```

```
#SBATCH --qos=urgent # Cola
#SBATCH --output=prueba_%j.log # Standard output and error log

echo "Hello World!"

hpc-login2 ~]$ sbatch trabajo_ejemplo.sh
```

Uso de los nodos con GPU

Para solicitar específicamente una asignación de GPUs para un trabajo hay que añadir a sbatch o srun las opciones:

--gres	Solicitud de gpus por NODE	--gres=gpu[:type]:count],...
--gpus o -G	Solicitud de gpus por JOB	--gpus=[type]:count,...

También existen las opciones --gpus-per-socket, --gpus-per-node y --gpus-per-task, Ejemplos:

```
## Ver la lista de nodos y gpus:
hpc-login2 ~]$ ver_recursos
## Solicitar 2 GPU cualesquiera para un JOB, añadir:
--gpus=2
## Solicitar una A100 de 40G en un nodo y una A100 de 80G en otro, añadir:
--gres=gpu:A100_40:1,gpu:A100_80:1
```

Monitorización de los trabajos

```
## Listado de todos los trabajos en la cola
hpc-login2 ~]$ squeue
## Listado de los trabajos de un usuario
hpc-login2 ~]$ squeue -u <login>
## Cancelar un trabajo:
hpc-login2 ~]$ scancel <JOBID>
## Lista de trabajos recientes
hpc-login2 ~]$ sacct -b
## Información histórica detallada de un trabajo:
hpc-login2 ~]$ sacct -l -j <JOBID>
## Información de debug de un trabajo para troubleshooting:
hpc-login2 ~]$ scontrol show jobid -dd <JOBID>
## Ver el uso de recursos de un trabajo en ejecución:
hpc-login2 ~]$ sstat <JOBID>
```

Controlar la salida de los trabajos

Códigos de salida

Por defecto estos son los códigos de salida de los comandos:

SLURM command	Exit code
salloc	0 en caso de éxito, 1 si no se puede ejecutar el comando del usuario
srun	El más alto de entre todas las tareas ejecutadas o 253 para un error out-of-mem
sbatch	0 en caso de éxito, si no, el código de salida correspondiente del proceso que falló

STDIN, STDOUT y STDERR

SRUN:

Por defecto stdout y stderr se redirigen de todos los TASKS a el stdout y stderr de srun, y stdin se redirecciona desde el stdin de srun a todas las TASKS. Esto se puede cambiar con:

-i, --input=<opcion>
-o, --output=<opcion>
-e, --error=<opcion>

Y las opciones son:

- *all*: opción por defecto.
- *none*: No se redirecciona nada.
- *taskid*: Solo se redirecciona desde y/o al TASK id especificado.
- *filename*: Se redirecciona todo desde y/o al fichero especificado.
- *filename pattern*: Igual que filename pero con un fichero definido por un [patrón](#)

SBATCH:

Por defecto “/dev/null” está abierto en el stdin del script y stdout y stderr se redirigen a un fichero de nombre “slurm-%j.out”. Esto se puede cambiar con:

-i, --input=<filename_pattern>
-o, --output=<filename_pattern>
-e, --error=<filename_pattern>

La referencia de filename_pattern está [aquí](#).

Envío de correos

Se pueden configurar los JOBS para que envíen correos en determinadas circunstancias usando estos dos parámetros (**SON NECESARIOS AMBOS**):

--mail-type=<type>	Opciones: BEGIN, END, FAIL, REQUEUE, ALL, TIME_LIMIT, TIME_LIMIT_90, TIME_LIMIT_50.
--mail-user=<user>	La dirección de correo de destino.

Estados de los trabajos en el sistema de colas

```
hpc-login2 ~]# squeue -l
```

JOBID	PARTITION	NAME	USER	STATE	TIME	NODES
-------	-----------	------	------	-------	------	-------

NODELIST (REASON)

6547 defaultPa example <username> RUNNING 22:54:55 1 hpc-fat1

Estados (STATE) más comunes de un trabajo:

- R RUNNING Job currently has an allocation.
- CD COMPLETED Job has terminated all processes on all nodes with an exit code of zero.
- F FAILED Job terminated with non-zero exit code or other failure condition.
- PD PENDING Job is awaiting resource allocation.

[Lista completa de posibles estados de un trabajo](#) .

Si un trabajo no está en ejecución aparecerá una razón debajo de REASON: [Lista de las razones](#) por las que un trabajo puede estar esperando su ejecución.

From:

<https://wiki.citius.usc.es/> - **Wiki do CiTIUS**

Permanent link:

<https://wiki.citius.usc.es/centro:servizos:hpc>

Last update: **2022/05/05 16:45**

