Centro Informático Científico de Andalucía
CONSEJERÍA DE INDUSTRIA, ENERGÍA Y MINAS

Manual de usuario para el uso del clúster Hércules del Centro Informático Científico de Andalucía (CICA)

Índice de contenido

1. Información del clúster

1.1. Introducción

En la era de la información y la innovación tecnológica, los supercomputadores se han convertido en herramientas esenciales para abordar problemas complejos y realizar cálculos de alta precisión en diversos campos de la ciencia, la ingeniería y la industria. Nuestro clúster de supercomputación, diseñado con tecnología de vanguardia, ofrece un rendimiento excepcional y una capacidad de procesamiento masiva, permitiendo a investigadores llevar a cabo simulaciones avanzadas, análisis de grandes volúmenes de datos y desarrollo de nuevas aplicaciones.

El clúster "***Hércules***" está compuesto por una red de nodos interconectados que trabajan en paralelo para ejecutar tareas a velocidades incomparables, optimizando el uso de recursos y asegurando la máxima eficiencia. Con una infraestructura robusta y escalable, nuestro sistema está preparado para soportar una amplia gama de aplicaciones, desde la modelización climática y la investigación genómica hasta el diseño de materiales y la inteligencia artificial.

Además, nuestro equipo de soporte técnico está dedicado a proporcionar asistencia continua, facilitando la transición y la optimización de aplicaciones para que los usuarios puedan aprovechar al máximo las capacidades del clúster.

Invitamos a investigadores, académicos y profesionales de todas las disciplinas a explorar las posibilidades que ofrece nuestro clúster de supercomputación, un recurso potente y versátil que impulsa el progreso científico y tecnológico.

1.2. Nodos del clúster

Tipo (Número de Nodos)	Hostname	CPU	Núcleos	RAM	Descripción
Login (3)	login[1-3].spc.cica.es	Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz	36	64 GB	Nodos de acceso
Cómputo (224)	nodo1[01-18][1-4].spc.cica.es nodo[3-4][01-19][1-4].spc.cica.es	2x Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz	96	183 GB DDR4	Nodos de cómputo
Cómputo – FAT (2)	fat[3-4]1.spc.cica.es	2x Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz	96	1506 GB DDR4	Nodos de cómputo FAT
GPU (6)	gpu1[1-4].spc.cica.es gpu[3-4]1.spc.cica.es	2x Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz	96	183 GB DDR4	Nodos de cómputo con tarjeta gráfica Nvidia A100 de 40 GB

El sistema operativo en el clúster Hércules es Rocky Linux 8.5 (Green Obsidian).

1.3. Gestión de datos – Sistema de Ficheros

En el clúster Hércules proporcionamos sistemas de archivos compartidos LUSTRE, ofreciendo tanto sistemas de archivos home como scratch, cada uno con propósitos específicos.

El directorio *$HOME* está destinado al almacenamiento de los datos del usuario y no cuentan con copias de seguridad. El uso del clúster se debe ceñir a almacenar en /home/usuario los ficheros estrictamente necesarios para ejecutar sus cálculos. Una vez obtenidos los resultados, descargárselos en su equipo personal liberando el espacio de /home/usuario.

A continuación, se presenta un resumen de los sistemas de archivos disponibles para los usuarios:

Sistema de Archivos	Punto de Montaje	Descripción
Lustre	/home	Sistemas de archivos home - sin copia de seguridad

Nota: Cada usuario tiene asignada una cuota de almacenamiento de **1 TB**.

1.4. Acceso al clúster

Para acceder al clúster, el usuario deberá rellenar el formulario de registro disponible en el siguiente enlace: https://www.cica.es/registro-hercules/.

Una vez completado el formulario, el usuario recibirá instrucciones detalladas para la instalación y configuración de la VPN del centro, la cual es necesaria para acceder a los recursos de supercomputación.

Con la VPN correctamente configurada y funcionando, el usuario podrá acceder a los nodos de entrada del clúster mediante SSH de la siguiente manera:

```
$ ssh usuario@login.spc.cica.es
```

El centro pone a disposición de los usuarios una plataforma web que facilita el uso del clúster mediante una interfaz gráfica, sin necesidad de emplear la línea de comandos.

Se puede acceder a dicha plataforma a través del siguiente enlace:

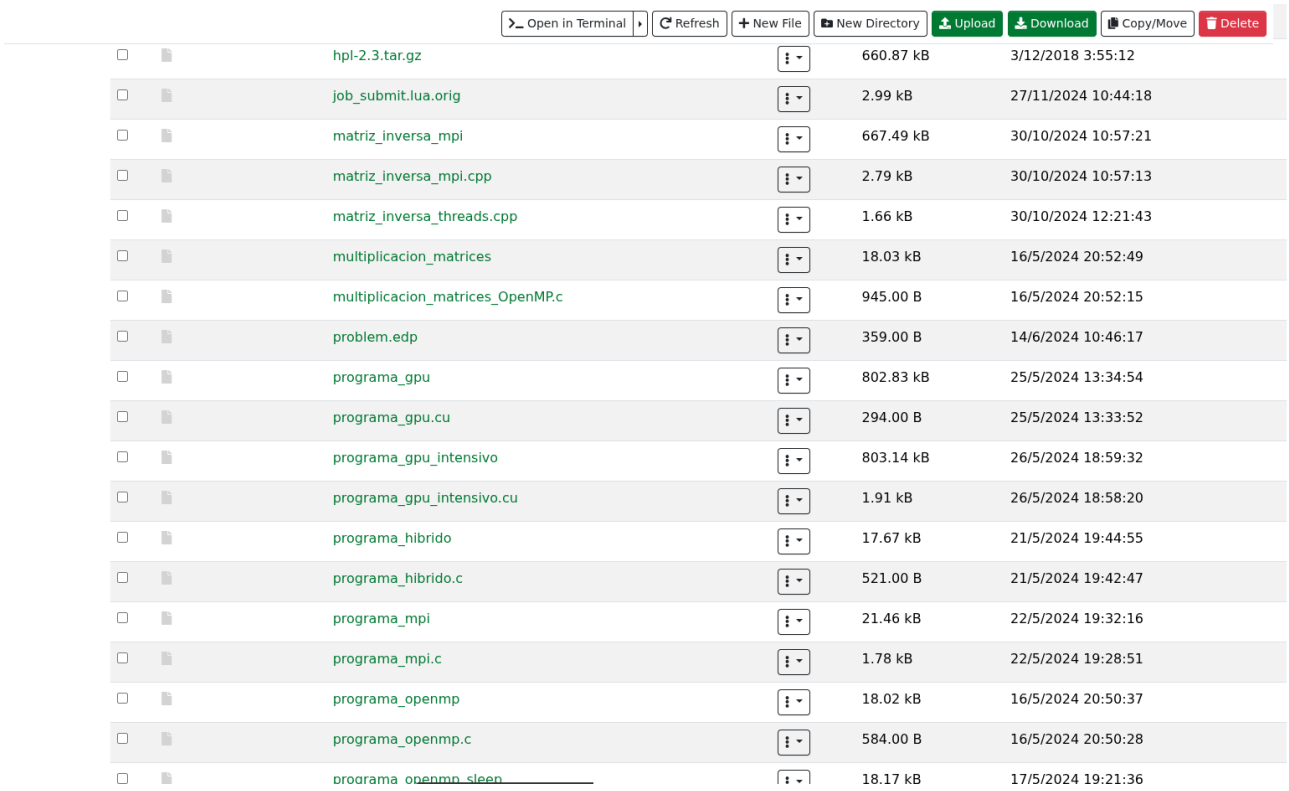
👉 https://ood-hercules.spc.cica.es

Para su correcto funcionamiento, es necesario tener activa la conexión VPN del centro y acceder con sus credenciales habituales del clúster.

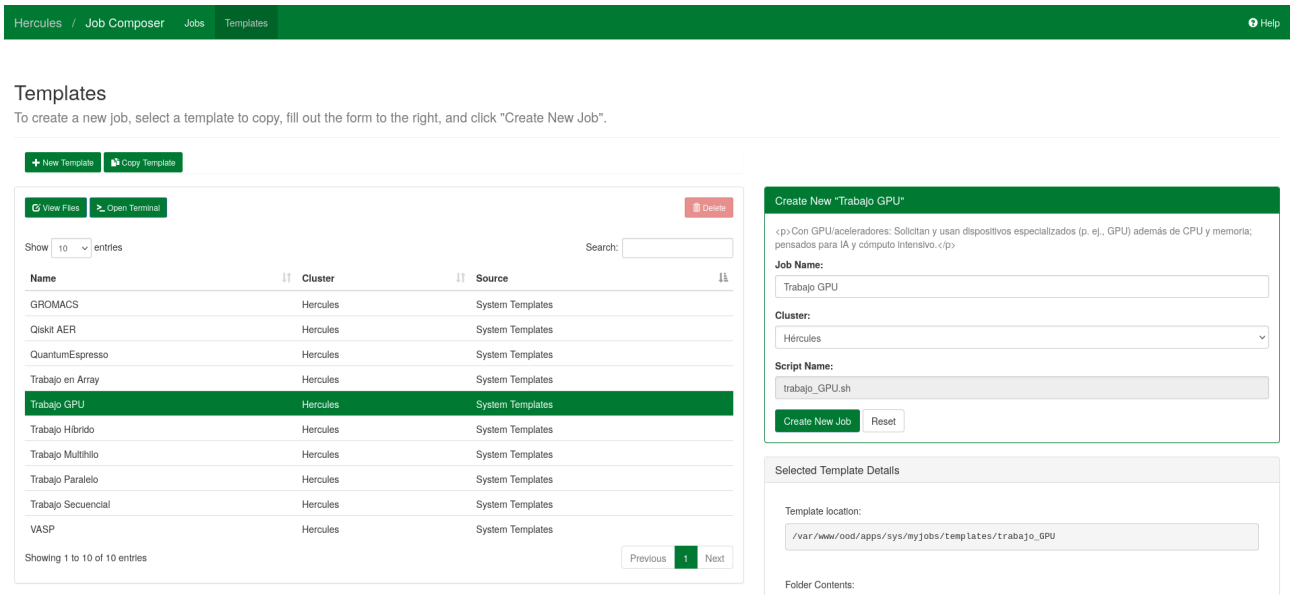
Esta herramienta permite gestionar trabajos, acceder a terminales interactivos, supervisar el uso de recursos y utilizar diversas aplicaciones científicas de manera más sencilla e intuitiva.

- Ejecución de aplicaciones interactivas:

- Intercambio de ficheros:

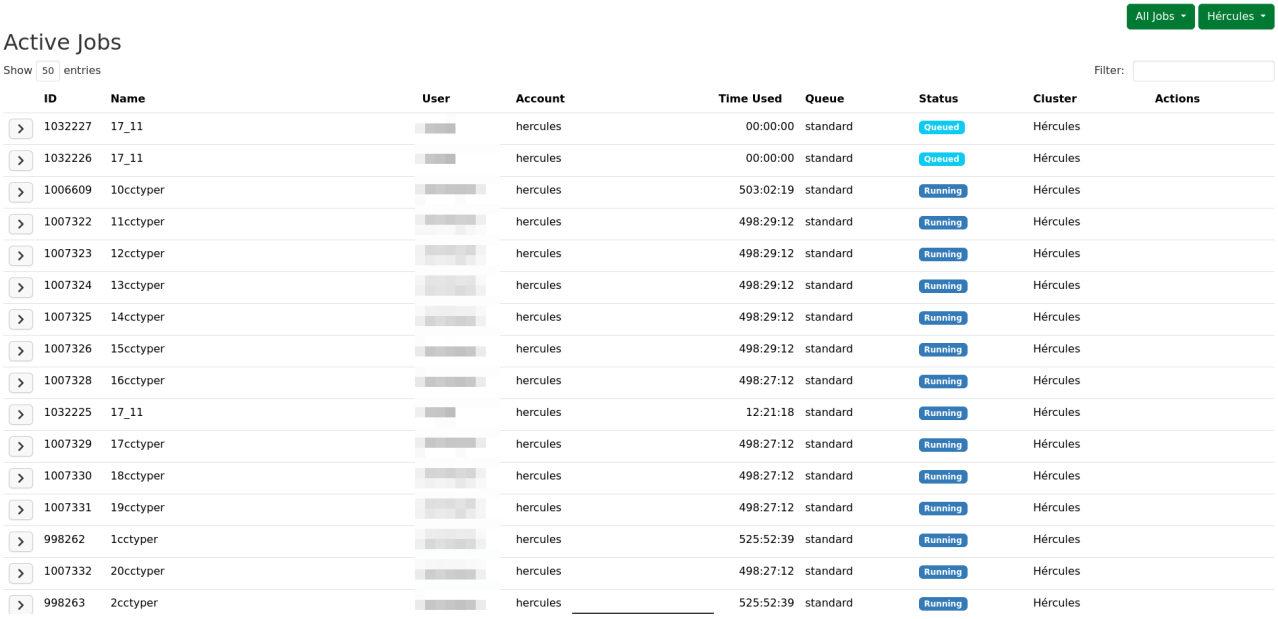


- Gestión de plantillas predefinidas y ejecución de trabajos:



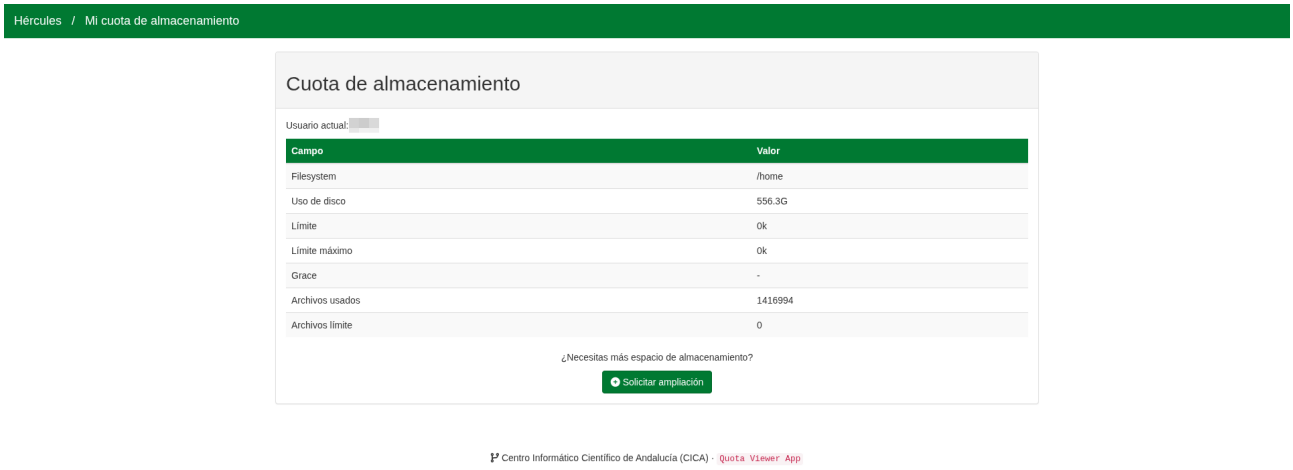
- Cambio de contraseña de la cuenta:

- Monitorización de los trabajos en curso:



	ID	Name	User	Account	Time Used	Queue	Status	Cluster	Actions
›	1032227	17_11	[redacted]	hercules	00:00:00	standard	Queued	Hércules	
›	1032226	17_11	[redacted]	hercules	00:00:00	standard	Queued	Hércules	
›	1006609	10cctyper	[redacted]	hercules	503:02:19	standard	Running	Hércules	
›	1007322	11cctyper	[redacted]	hercules	498:29:12	standard	Running	Hércules	
›	1007323	12cctyper	[redacted]	hercules	498:29:12	standard	Running	Hércules	
›	1007324	13cctyper	[redacted]	hercules	498:29:12	standard	Running	Hércules	
›	1007325	14cctyper	[redacted]	hercules	498:29:12	standard	Running	Hércules	
›	1007326	15cctyper	[redacted]	hercules	498:29:12	standard	Running	Hércules	
›	1007328	16cctyper	[redacted]	hercules	498:27:12	standard	Running	Hércules	
›	1032225	17_11	[redacted]	hercules	12:21:18	standard	Running	Hércules	
›	1007329	17cctyper	[redacted]	hercules	498:27:12	standard	Running	Hércules	
›	1007330	18cctyper	[redacted]	hercules	498:27:12	standard	Running	Hércules	
›	1007331	19cctyper	[redacted]	hercules	498:27:12	standard	Running	Hércules	
›	998262	1cctyper	[redacted]	hercules	525:52:39	standard	Running	Hércules	
›	1007332	20cctyper	[redacted]	hercules	498:27:12	standard	Running	Hércules	
›	998263	2cctyper	[redacted]	hercules	525:52:39	standard	Running	Hércules	

- Gestión y ampliación de la cuota de almacenamiento:



1.5. Módulos

El software instalado en Hércules está organizado mediante una jerarquía de módulos. Al cargar un módulo, se adaptan las variables de entorno para proporcionar acceso a un conjunto específico de software y sus dependencias. Esta organización jerárquica de los módulos garantiza que se obtenga un conjunto coherente de dependencias, como por ejemplo, todas construidas con la misma versión del compilador o todas basadas en la misma implementación de MPI (Message Passing Interface).

1.5.1. Uso de los módulos

Los usuarios deben cargar, descargar y consultar módulos a través del comando module. Algunos comandos útiles son:

Comando	Descripción
`module avail`	Muestra los módulos disponibles.
`module load <nombre_del_módulo>/<versión_del_módulo>`	Carga un módulo específico. Se carga la versión predeterminada si no se especifica una versión.
`module list`	Lista los módulos que están actualmente cargados.
`module unload <nombre_del_módulo>/<versión_del_módulo>`	Descarga un módulo.
`module purge`	Descarga todos los módulos que hayan sidos cargados previamente.

Comando	Descripción
`module spider <nombre_del_módulo>`	Encuentra la ubicación de un módulo dentro de la jerarquía de módulos.

A continuación se muestran algunos ejemplos del uso de los módulos

- **Listar los módulos disponibles en el clúster:**

```
$ module avail

------------------------------------------------------------------------
/lustre/software/easybuild/x86_64/modules/all
------------------------------------------------------------------------
   AOCL/3.1.0                                                M4/1.4.18
expat/2.4.1-GCCcore-11.2.0
   ATLAS/3.10.2-GCC-11.2.0-LAPACK-3.6.1                      M4/1.4.19-GCCcore-
11.2.0                              expat/2.4.8-GCCcore-11.3.0
(D)
   Anaconda3/2022.05                                         M4/1.4.19-GCCcore-
11.3.0                              flex/2.6.3
   Arrow/6.0.0-foss-2021b                                    M4/1.4.19
(D)    flex/2.6.4-GCCcore-11.2.0
   Autoconf/2.71-GCCcore-11.2.0                              MCL/14.137-GCCcore-
11.2.0                             flex/2.6.4-GCCcore-11.3.0
   Automake/1.16.4-GCCcore-11.2.0                            METIS/5.1.0-
GCCcore-11.2.0                           flex/2.6.4
(D)
   Autotools/20210726-GCCcore-11.2.0                         MMseqs2/13-45111-
gompi-2021b                          fontconfig/2.13.94-GCCcore-11.2.0
   BLAST+/2.2.30                                             MPFR/4.1.0-GCCcore-
11.2.0                             foss/2021b
...
```

- **Buscar un módulo:**

```
$ module spider paraview

------------------------------------------------------------------------------
------------------------------------------------------------------------------
----------------------------
  ParaView: ParaView/5.9.1-foss-2021b-mpi
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----------------------------
    Description:
      ParaView is a scientific parallel visualizer.


     This module can be loaded directly: module load ParaView/5.9.1-foss-
2021b-mpi
```

```
   Help:

     Description
     ===========
     ParaView is a scientific parallel visualizer.


     More information
     ================
      - Homepage: https://www.paraview.org
```

- **Cargar un módulo:**

```
$ module load ParaView/5.9.1-foss-2021b-mpi
```

- **Mostrar los módulos cargados:**

```
$ module list

Currently Loaded Modules:
   1) GCCcore/11.2.0                          17) FlexiBLAS/3.0.4-GCC-11.2.0
33) Szip/2.1.1-GCCcore-11.2.0           49) xorg-macros/1.19.3-GCCcore-11.2.0
      2)   zlib/1.2.11-GCCcore-11.2.0                      18)   gompi/2021b
34) HDF5/1.12.1-gompi-2021b             50) X11/20210802-GCCcore-11.2.0
    3)  binutils/2.37-GCCcore-11.2.0           19)  FFTW/3.3.10-gompi-2021b
35) cURL/7.78.0-GCCcore-11.2.0          51) FriBidi/1.0.10-GCCcore-11.2.0
  4) GCC/11.2.0                           20) ScaLAPACK/2.1.0-gompi-2021b-fb
36) netCDF/4.8.1-gompi-2021b            52) FFmpeg/4.3.2-GCCcore-11.2.0
       5)    numactl/2.0.14-GCCcore-11.2.0                 21)    foss/2021b
37) GSL/2.7-GCC-11.2.0                  53) Voro++/0.4.6-GCCcore-11.2.0
   6)  XZ/5.2.5-GCCcore-11.2.0                22)  bzip2/1.0.8-GCCcore-11.2.0
38) gzip/1.10-GCCcore-11.2.0            54) libarchive/3.5.1-GCCcore-11.2.0
    7)  libxml2/2.9.10-GCCcore-11.2.0         23)  ncurses/6.2-GCCcore-11.2.0
39) tbb/2020.3-GCCcore-11.2.0           55) CMake/3.21.1-GCCcore-11.2.0
   8)  libpciaccess/0.16-GCCcore-11.2.0   24)  libreadline/8.1-GCCcore-11.2.0
40) PCRE/8.45-GCCcore-11.2.0            56) kim-api/2.2.1-GCCcore-11.2.0
    9)  hwloc/2.5.0-GCCcore-11.2.0             25)  Tcl/8.6.11-GCCcore-11.2.0
41) x264/20210613-GCCcore-11.2.0        57) Eigen/3.3.9-GCCcore-11.2.0
 10)  OpenSSL/1.1                             26)  SQLite/3.36-GCCcore-11.2.0
42) LAME/3.100-GCCcore-11.2.0           58) pybind11/2.7.1-GCCcore-11.2.0
  11)  libevent/2.1.12-GCCcore-11.2.0         27)  GMP/6.2.1-GCCcore-11.2.0
43) x265/3.5-GCCcore-11.2.0             59) SciPy-bundle/2021.10-foss-2021b
 12)  UCX/1.11.2-GCCcore-11.2.0              28)  libffi/3.4.2-GCCcore-11.2.0
44) expat/2.4.1-GCCcore-11.2.0          60) ICU/69.1-GCCcore-11.2.0
  13)  libfabric/1.13.2-GCCcore-11.2.0      29)  Python/3.9.6-GCCcore-11.2.0
45) Brotli/1.0.9-GCCcore-11.2.0         61) Boost/1.77.0-GCC-11.2.0
 14)  PMIx/4.1.0-GCCcore-11.2.0              30)  libpng/1.6.37-GCCcore-11.2.0
46) freetype/2.11.0-GCCcore-11.2.0      62) PLUMED/2.7.3-foss-2021b
 15)  OpenMPI/4.1.1-GCC-11.2.0               31)  NASM/2.15.05-GCCcore-11.2.0
47) util-linux/2.37-GCCcore-11.2.0      63) ScaFaCoS/1.0.1-foss-2021b
 16)  OpenBLAS/0.3.18-GCC-11.2.0             32)  libjpeg-turbo/2.0.6-GCCcore-
```

```
11.2.0   48) fontconfig/2.13.94-GCCcore-11.2.0   64) LAMMPS/29Sep2021-foss-
2021b-kokkos
```

- **Descargar todos los módulos cargados previamente:**

```
$ module purge
```

2. Gestor de Colas (SLURM)

2.1. Particiones

En Slurm, los nodos pueden agruparse en particiones, las cuales son conjuntos de nodos con límites específicos asociados, tales como el tiempo de reloj y el tamaño del trabajo, entre otros. Estos límites son estrictos para los trabajos y no pueden ser anulados por los límites especificados en las Políticas de Calidad de Servicio (QoS). Las particiones pueden superponerse, y los nodos pueden pertenecer a más de una partición, permitiendo que las particiones actúen como colas de propósito general.

A continuación, se presenta una tabla que muestra las particiones en Hércules, junto con los límites máximos configurados y los valores predeterminados:

Partición	Límite	Valor
`standard (predeterminada)`	Número mínimo/máximo de nodos por trabajo	1 / 232 nodos
	Número máximo de tareas en ejecución por nodo	96 tareas
`gpu`	Número mínimo/máximo de nodos por trabajo	1/6
	Número máximo de gpus disponibles por nodo	1

3. Comandos de usuario para SLURM

En esta sección daremos primero una lista de todos los comandos con una breve descripción y después describiremos con más detalle la funcionalidad de cada comando, dando también algunos ejemplos.

3.1. Comandos para lanzar trabajos

Slurm ofrece una variedad de comandos de usuario para todas las acciones necesarias relativas a los trabajos. Con estos comandos los usuarios tienen una interfaz rica para asignar recursos, consultar el estado de los trabajos, controlar los

trabajos, gestionar información contable y simplificar su trabajo con algunos comandos de utilidad.

A continuación, se detalla la lista de comandos de usuario disponibles en Slurm, así como su respectiva descripción y funcionalidad:

- **salloc**: Se utiliza para solicitar trabajos o asignaciones interactivas. Al iniciar el trabajo, se abre un shell (u otro programa especificado en la línea de comandos) en el nodo de envío (nodo de inicio de sesión). Desde este shell, se puede utilizar el comando **srun** para ejecutar aplicaciones paralelas de forma interactiva. La asignación se libera cuando el usuario cierra el shell.
- **sattach**: Permite adjuntar las capacidades de entrada, salida y error estándar, además de señales, a un trabajo en ejecución. Es posible adjuntar y desadjuntar de los trabajos múltiples veces.
- **sbatch**: Sirve para enviar un script por lotes. El script se ejecutará en el primer nodo de la asignación seleccionada por el planificador, y el directorio de trabajo coincidirá con el del comando **sbatch**. Dentro del script, se pueden utilizar uno o varios comandos **srun** para ejecutar aplicaciones paralelas (MPI).
- **scancel**: Se emplea para cancelar un trabajo pendiente o en ejecución.
- **sbcast**: Utilizado para transferir un archivo a todos los nodos asignados a un trabajo. Este comando solo puede utilizarse dentro de un script de trabajo.
- **sgather**: Permite transferir un archivo desde todos los nodos asignados al trabajo actualmente activo. Al igual que **sbcast**, este comando solo puede usarse dentro de un script de trabajo.
- **sinfo**: Proporciona información sobre las particiones, reservas y estados de los nodos. Ofrece una amplia variedad de opciones de filtrado, clasificación y formato.
- **sprio**: Permite consultar las prioridades de los trabajos.
- **squeue**: Permite consultar la lista de trabajos pendientes y en ejecución. Por defecto, muestra la lista de trabajos pendientes ordenados por prioridad y la lista de trabajos en ejecución también ordenados por prioridad.
- **srun**: Utilizado para iniciar pasos de trabajo dentro de un trabajo o para comenzar un trabajo interactivo. Este comando ofrece diversas opciones para especificar los requisitos de recursos. Un trabajo puede contener múltiples pasos de trabajo que se ejecutan secuencialmente o en paralelo en nodos independientes o compartidos dentro de la asignación de nodos del trabajo.

- **sstat**: Facilita la consulta del estado de un trabajo en ejecución, proporcionando información detallada sobre su progreso.
- **sacct**: Se utiliza para recuperar información contable sobre trabajos y pasos de trabajo. Este comando es esencial para la consulta de datos históricos sobre la ejecución de trabajos.

Nota: Existen páginas de manual disponibles para todos los demonios, comandos y funciones API. La opción de comando "--help" proporciona un resumen breve de las opciones disponibles, facilitando así el uso eficiente de estos comandos.

3.2. Comandos para el control de los trabajos

3.2.1. scancel

Con *scancel*, podemos enviar señales o cancelar trabajos, matrices de trabajos (arrays) o pasos de trabajos.

Formato del comando:

```
scancel [OPCIONES...] [job_id[,array_id][.step_id]...]
```

Algunas de las opciones más útiles del comando *scancel* son:

Opción	Descripción
`-A <cuenta>`	Restringe la operación solo a los trabajos bajo la cuenta especificada.
`-b --batch`	Envía una señal al shell del trabajo por lotes y a sus procesos hijos.
`-i --interactive`	Habilita el modo interactivo. El usuario debe confirmar para cada operación.
`-n <nombre_trabajo>`	Cancela un trabajo con el nombre especificado.
`-p <nombre_partición>`	Restringe la operación solo a los trabajos que se ejecutan en la partición especificada.
`-R <nombre_reserva>`	Restringe la operación solo a los trabajos que se ejecutan usando la reserva especificada.
`-s <nombre_señal>`	Envía una señal al(los) trabajo(s) especificado(s).

Opción	Descripción
`-t <nombre_estado_trabajo>`	Restringe la operación solo a los trabajos que tienen el estado especificado. Por favor, consulta la página del manual.
`-u <nombre_usuario>`	Cancela trabajo(s) solo del usuario especificado. Si no se da ninguna ID de trabajo, se cancelarán todos los trabajos de este usuario.

Ejemplos:

- Cancelar trabajos con ID 777 y 778:

```
scancel 777 778
```

- Cancelar trabajos con los nombres especificados:

```
scancel -n testjob1 testjob2
```

- Cancelar todos los trabajos en cola (pendientes, en ejecución, etc.) del usuario1:

```
scancel -u user1
```

- Cancelar todos los trabajos en la partición large que pertenecen al usuario1:

```
scancel -p large -u user1
```

- Cancelar todos los trabajos del usuario1 que están en estado pendiente:

```
scancel -t PENDING -u user1
```

3.3. Comandos para monitorización de los trabajos

3.3.1. squeue

Con el comando *squeue*, podemos ver la información actual del estado de los trabajos en cola y en ejecución.

Formato del comando:

```
squeue [OPCIONES...]
```

Algunas de las opciones más útiles de *squeue* son:

Opción	Descripción
`-A <account_list>` `--account=<account_list>`	Lista los trabajos para las cuentas especificadas.
`-a` `--all`	Muestra información sobre trabajos y pasos de trabajo para todas las particiones.
`-r` `--array`	Visualización optimizada para matrices de trabajos.
`-h` `--noheader`	No imprime el encabezado de la salida.
`-i <seconds>` `--iterate=<seconds>`	Imprime repetidamente la información en el intervalo especificado.
`-l` `--long`	Reporta más información.
`-o <output_format>` `--format=<output_format>`	Especifica la información que se imprimirá (columnas). Por favor, lea las páginas del manual para más información.
`-p <part_list>` `--partition=<part_list>`	Lista trabajos solo de las particiones especificadas.
`-R <reservation_name>` `--reservation=<reservation_name>`	Lista trabajos solo para la reserva especificada.
`-S <sort_list>` `--sort=<sort_list>`	Especifica el orden de los trabajos listados.
`--start`	Imprime el tiempo estimado de inicio para cada trabajo en la cola.
`-t <state_list>` `--states=<state_list>`	Lista trabajos solo con el estado especificado (fallido, pendiente, en ejecución, etc.).
`-u <user_list>` `--user=<user_list>`	Imprime los trabajos del usuario especificado.

Ejemplos:

- Imprimir repetidamente el estado de la cola cada 4 segundos:

```
squeue -i 4
```

- Mostrar trabajos en la partición gpu:

```
squeue -p gpu
```

- Mostrar trabajos que pertenecen a un usuario específico:

```
squeue -u user01
```

- Imprimir el estado de la cola con un formato personalizado, mostrando solo ID del trabajo, partición, usuario y estado del trabajo:

```
squeue --format="%.18i %.9P %.8u %.2t"
```

Estados comunes de los trabajos en Slurm:

Normalmente, los trabajos pasarán por varios estados durante su ciclo de vida. Los estados típicos de los trabajos desde la presentación hasta la finalización son: PENDIENTE (PD), EN EJECUCIÓN (R), COMPLETANDO (CG) y COMPLETADO (CD). Sin embargo, hay muchos posibles estados de trabajo en Slurm. La siguiente tabla describe los estados más comunes:

Código del Estado	Nombre del Estado	Descripción
CA	CANCELADO	El trabajo fue cancelado explícitamente por el usuario o un administrador. El trabajo puede o no haberse iniciado.
CD	COMPLETADO	El trabajo ha terminado todos los procesos en todos los nodos.
CF	CONFIGURANDO	Al trabajo se le han asignado recursos, pero está esperando que se preparen para su uso.
CG	COMPLETANDO	El trabajo está en proceso de completarse. Algunos procesos en algunos nodos aún pueden estar activos. Usualmente, Slurm está descargando la salida del trabajo durante este estado.
F	FALLIDO	El trabajo terminó con un código de salida diferente a cero u otra condición de falla.
NF	FALLO DE NODO	El trabajo terminó debido a la falla de uno o más nodos asignados.
PD	PENDIENTE	El trabajo está esperando la asignación de recursos.
R	EJECUTÁNDOSE	El trabajo actualmente tiene una asignación. Nota: Slurm siempre ejecuta el prólogo al comienzo de cada trabajo antes de la ejecución real de la aplicación del

Código del Estado	Nombre del Estado	Descripción
		usuario.
TO	TIEMPO EXCEDIDO	El trabajo terminó al alcanzar su límite de tiempo

3.3.2. sstat

Con el comando *sstat*, podemos obtener diversa información de estado sobre los trabajos en ejecución. Este comando proporciona valores mínimos, máximos y promedios para diversas métricas, tales como el tiempo de CPU, el uso de memoria virtual, la E/S de disco, el número de tareas, entre otros. Estas métricas permiten a los usuarios monitorear y analizar el rendimiento de sus trabajos, facilitando la optimización de los recursos utilizados.

Formato del comando:

```
sstat [OPCIONES...]
```

Algunas de las opciones más útiles de *sstat* son:

Opción	Descripción
`-a` `--allsteps`	Muestra información sobre todos los pasos para el trabajo especificado.
`-e` `--helpformat`	Muestra la lista de campos que se pueden especificar con la opción --format.
`-i` `--pidformat`	Muestra información sobre los PID de cada paso de trabajo.
`-j <trabajo[.paso]>` `--jobs <trabajo[.paso]>`	Muestra información para los trabajos o pasos de trabajo especificados.
`-n` `--noheader`	No muestra el encabezado al inicio de la salida.

`-o <lista_campos>`
`--format=<lista_campos>`
`--fields=<lista_campos>`
	Especifica la lista de campos separados por comas que se mostrarán en la salida. Los campos disponibles se pueden encontrar con la opción -e o en las páginas del manual.

`-p --parsable` `-P --parsable2`	Imprime información en un formato analizable. La salida se delimitará con `

Ejemplos:

- Mostrar la información de estado predeterminada para el trabajo 777:

```
sstat -j 777
```

- Mostrar las métricas definidas para el trabajo 777 en formato analizable:

```
sstat -P --format=JobID,AveCPU,AvePages,AveRSS,AveVMSize -j 777
```

3.3.3. sacct

Con el comando *sacct*, podemos obtener información y datos de contabilidad de los trabajos que están almacenados en la base de datos de contabilidad de Slurm. Slurm guarda el historial de todos los trabajos en esta base de datos, permitiendo a los usuarios acceder a un registro detallado de sus trabajos anteriores. Sin embargo, cada usuario tiene permisos únicamente para verificar sus propios trabajos, garantizando la privacidad y seguridad de la información de los demás usuarios. Este comando es esencial para el seguimiento y análisis del rendimiento y uso de recursos de los trabajos ejecutados.

Formato del comando:

```
sacct [OPCIONES...]
```

Algunas de las opciones más útiles de *sacct* son:

Opción	Descripción
`-b --brief`	Muestra una lista breve, con los campos: jobid, status y exitcode.
`-e --helpformat`	Muestra la lista de campos que se pueden especificar con la opción --format.
`-E <hora_fin> --endtime=<hora_fin>`	Lista trabajos con cualquier estado (o con estados especificados usando la opción --state) antes de la fecha dada. Por favor consulte las páginas del manual para los formatos de tiempo disponibles.
`-j <trabajo(.paso)> --jobs=<trabajo(.paso)>`	Muestra información solo para los trabajos/pasos de trabajo especificados.

Opción	Descripción
`-l --long`	Muestra un informe completo con todos los campos disponibles para cada trabajo/paso de trabajo reportado.
`-n --noheader`	No muestra el encabezado al inicio de la salida.
`-N <lista_nodos> --nodelist=<lista_nodos>`	Muestra información solo para trabajos que se ejecutaron en los nodos especificados.
`--name=<lista_nombres_trabajo>`	Muestra información sobre trabajos con los nombres especificados.
`-o <lista_campos> --format=<lista_campos>`	Especifica la lista de campos que se mostrarán en la salida. Los campos disponibles se pueden encontrar con la opción -e o en las páginas del manual.
`-r <nombre_partición> --partition=<nombre_partición>`	Muestra información solo para trabajos que se ejecutaron en las particiones especificadas. El valor predeterminado son todas las particiones.
`-s <lista_estados> --state=<lista_estados>`	Filtra y muestra información solo sobre trabajos con los estados especificados, como completados, cancelados, fallidos, etc. Por favor consulte las páginas del manual para la lista completa de estados.
`-S <hora_inicio> --starttime=<hora_inicio>`	Lista trabajos con cualquier estado (o con estados especificados usando la opción --state) después de la fecha dada. El valor predeterminado es 00:00:00 de la fecha actual. Consulte las páginas del manual para los formatos de fecha.
`-X --allocations`	Muestra información solo para trabajos y no para pasos de trabajo.

Ejemplos:

- Mostrar información de trabajos en formato largo para el período predeterminado (desde las 00:00 de hoy hasta ahora):

```
sacct -l
```

- Mostrar información solo de trabajos (sin jobsteps) desde la fecha definida

hasta ahora:

```
sacct -S 2024-10-01T07:33:00 -X
```

- Mostrar información de trabajos y jobsteps imprimiendo solo los campos especificados:

```
sacct -S 2024-10-01 --format=jobid,elapsed,nnodes,state
```

3.3.4. sinfo

Con *sinfo*, podemos obtener información y verificar el estado actual de las particiones, nodos y reservas. Este comando es útil para comprobar la disponibilidad de los nodos.

Formato del comando:

```
sinfo [OPCIONES...]
```

Algunas de las opciones más útiles de *sinfo* son:

Opción	Descripción
`-a` `--all`	Muestra información sobre todas las particiones.
`-d` `--dead`	Muestra información solo para los nodos que no responden (muertos).
`-i <segundos>` `--iterate=<segundos>`	Imprime información repetidamente a intervalos especificados.
`-l --long`	Reporta más información.
`-n <nodos>` `--nodes=<nodos>`	Muestra información solo sobre los nodos especificados.
`-N --Node`	Muestra información en un formato orientado a nodos.
`-o <formato_salida>` `--format=<formato_salida>`	Especifica la información que se imprimirá (columnas). Consulte las páginas del manual para más información.
`-p <partición>` `--partition=<partición>`	Muestra información en un formato orientado a nodos.
`-r` `--responding`	Muestra información solo para los nodos que

Opción	Descripción
	responden.
`-R` `--list-reasons`	Lista las razones por las cuales los nodos no están en un estado saludable.
`-s --summarize`	Lista las particiones sin muchos detalles de los nodos.
`-t <estados>` `--states=<estados>`	Lista nodos solo con el estado especificado (por ejemplo, allocated, down, drain, idle, maint, etc.).
`-T` `--reservation`	Muestra información sobre las reservas.

Ejemplos:

- Mostrar información sobre nodos en estado inactivo:

```
sinfo -t idle
```

- Mostrar información sobre particiones y nodos de manera resumida:

```
sinfo -s
```

- Listar todas las reservas:

```
sinfo -R
```

- Mostrar trabajos que pertenecen a un usuario específico:

```
sinfo -T
```

- Mostrar información para la partición "gpu":

```
sinfo -p gpu
```

Dependiendo de las opciones, el comando **sinfo** imprimirá los estados de las particiones y los nodos. Las particiones pueden estar en estado **UP**, **DOWN** o **INACTIVE**. El estado **UP** significa que una partición aceptará nuevas solicitudes y los trabajos serán programados. El estado **DOWN** permite solicitudes a una partición, pero los trabajos no serán programados. El estado **INACTIVE** significa que no se permiten solicitudes.

Los nodos también pueden estar en varios estados. El código del estado del nodo puede abreviarse según el tamaño del campo impreso. La siguiente tabla muestra los estados de nodos más comunes:

Estado Abreviado	Nombre del Estado	Descripción
alloc	ALLOCATED	El nodo ha sido asignado.
comp	COMPLETING	El trabajo asociado con este nodo está en el estado de COMPLETANDO.
down	DOWN	El nodo no está disponible para su uso.
drain	DRAINING & DRAINED	Mientras está en estado DRAINING, cualquier trabajo en ejecución en el nodo se permitirá hasta su finalización. Después de eso y en estado DRAIN, el nodo no estará disponible para su uso.
idle	IDLE	El nodo no está asignado a ningún trabajo y está disponible para su uso.
maint	MAINT	El nodo está actualmente en una reserva con una bandera de "mantenimiento".
resv	RESERVED	El nodo está en una reserva avanzada y no está generalmente disponible.

4. Envío de trabajos por Lotes con SLURM

Los usuarios envían trabajos por lotes (generalmente scripts bash) utilizando el comando *sbatch*. En estos scripts de trabajo, para definir los parámetros de sbatch, se deben usar las directivas #SBATCH.

Para ejecutar tareas paralelas MPI, los usuarios llaman a *srun* dentro de su script. Con srun, los usuarios también pueden crear "job-steps". Un "job-step" puede asignar todos o un subconjunto de los recursos ya asignados por sbatch. Mediante estos comandos, SLURM ofrece un mecanismo para asignar recursos durante un cierto tiempo límite y luego ejecutar múltiples trabajos paralelos dentro de ese marco de tiempo.

A continuación, se presenta una tabla que describe las opciones de asignación más comunes o necesarias que se pueden definir en un script de trabajo:

Opción	Valor por defecto	Descripción
`#SBATCH --nodes=<número>`	1	Número de nodos para la asignación.
`#SBATCH -N <número>`	1	Número de nodos para la asignación.

Opción	Valor por defecto	Descripción
`#SBATCH --ntasks=<número>`	1	Número de tareas (procesos MPI). Se puede omitir si se dan --nodes y --ntasks-per-node.
`#SBATCH -n <número>`	1	Número de tareas (procesos MPI). Se puede omitir si se dan --nodes y --ntasks-per-node.
`#SBATCH --ntasks-per-node=<número>`	1	Número de tareas por nodo. Si se omite, se usa el valor predeterminado.
`#SBATCH --cpus-per-task=<número>`	1	Número de hilos/VCore por tarea. Utilizado solo para trabajos OpenMP o híbridos.
`#SBATCH -c <número>`	1	Número de hilos/VCore por tarea. Utilizado solo para trabajos OpenMP o híbridos.
`#SBATCH --output=<ruta>`	slurm-<jobID>.out	Ruta al archivo para la salida estándar.
`#SBATCH -o <ruta>`	slurm-<jobID>.out	Ruta al archivo para la salida estándar.
`#SBATCH --error=<ruta>`	slurm-<jobID>.out	Ruta al archivo para el error estándar.
`#SBATCH -e <ruta>`	slurm-<jobID>.out	Ruta al archivo para el error estándar.
`#SBATCH --time=<walltime>`	Obligatorio	Límite de tiempo solicitado para el trabajo.
`#SBATCH -t<walltime>`	Obligatorio	Límite de tiempo solicitado para el trabajo.
`#SBATCH --partition=<nombre>`	batch	Partición para ejecutar el trabajo.
`#SBATCH -p <nombre>`	batch	Partición para ejecutar el trabajo.
`#SBATCH --job-name=<nombretrabajo>`	nombre del script	Nombre del trabajo.
`#SBATCH -J <nombretrabajo>`	nombre del script	Nombre del trabajo.

Múltiples llamadas a srun pueden colocarse en un solo script por lotes. Opciones como `--nodes`, `--ntasks` y `--ntasks-per-node` se toman por defecto de los argumentos de sbatch, pero pueden ser sobrescritas para cada invocación de srun.

Como se describió anteriormente, el script de trabajo se envía utilizando:

```
sbatch <script_de_trabajo>
```

En caso de éxito, sbatch escribe el ID del trabajo en la salida estándar.

4.1. Ejemplos de Jobs scripts

4.1.1. Trabajos secuenciales

Ejemplo 1: Aquí hay un ejemplo simple donde se ejecuta un programa secuncial dentro del script de trabajo. Este trabajo tendrá el nombre "`trabajo_secuencial`". Se asignará un nodo de cómputo durante 1 hora y un sólo núcleo de CPU. La salida se escribirá en los archivos definidos. El trabajo se ejecutará en la partición predeterminada standard.

```
#!/bin/bash

#SBATCH --job-name=trabajo_secuencial    # Nombre del trabajo
#SBATCH --output=%x_%j.out               # Archivo de salida
#SBATCH --error=%x_%j.out                # Archivo de errores
#SBATCH --partition=standard             # Partición/cola
#SBATCH --nodes=1                        # 1 nodo
#SBATCH --ntasks=1                       # 1 tarea/proceso (secuencial)
#SBATCH --cpus-per-task=1                # 1 CPU para esa tarea
#SBATCH --mem=2G                         # Memoria RAM
#SBATCH --time=00:10:00                  # Tiempo máximo de ejecución
                                           (horas:minutos:segundos)

# Cargar un módulo (ajustar a disponibilidad del clúster)
module load Python/3.11.5-GCCcore-13.2.0

echo "==> Inicio en: $(date)"
echo "==> JobID: ${SLURM_JOB_ID:-N/A}"
echo "==> Directorio de envío: ${SLURM_SUBMIT_DIR:-$PWD}"

# Trabajo secuencial: un solo proceso ejecutando un comando
python -c 'import sys; print("Hola desde SLURM con Python");
print(sys.version)'

echo "==> Fin en: $(date)"
```

4.1.2. Trabajos pararelos

4.1.2.1. Trabajos OpenMP (Mútiples cores y memoria compartida)

Ejemplo 2: En este ejemplo, el trabajo ejecutará una aplicación OpenMP llamada "hello_omp". La asignación es para 1 nodo y se usarán 8 núcleos de CPU. También se definen los nombres de los archivos de salida y se solicita un tiempo de ejecución de 1 hora.

Nota: Es importante definir y exportar la variable *OMP_NUM_THREADS* que será utilizada por el ejecutable.

```
#!/bin/bash

#SBATCH --job-name=trabajo_multihilo      # Nombre del trabajo
#SBATCH --output=%x_%j.out                # Archivo de salida
#SBATCH --error=%x_%j.out                 # Archivo de errores
#SBATCH --partition=standard              # Partición/cola
#SBATCH --nodes=1                         # 1 nodo
#SBATCH --ntasks=1    # 1 tarea/proceso (OpenMP es multihilo dentro de 1 tarea)
#SBATCH --cpus-per-task=8                    # Nº de hilos que podrá usar OpenMP
#SBATCH --mem=2G                             # Memoria RAM
#SBATCH --time=00:01:00                      # Tiempo máximo de ejecución
                                               (horas:minutos:segundos)

# Cargar un módulo (ajustar a disponibilidad del clúster)
module load GCC/14.2.0

echo "==> Inicio en: $(date)"
echo "==> JobID: ${SLURM_JOB_ID:-N/A}"
echo "==> Directorio de envío: ${SLURM_SUBMIT_DIR:-$PWD}"
echo "==> Hilos OpenMP: ${OMP_NUM_THREADS}"

# Ajustar variables de OpenMP según los recursos asignados por SLURM
export OMP_NUM_THREADS="${SLURM_CPUS_PER_TASK}"

# Pequeño programa OpenMP de ejemplo (se compila y ejecuta)
cat > hello_omp.c <<'EOF'
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(void) {
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int nthreads = omp_get_num_threads();
        printf("Hola desde hilo %d de %d en nodo %s\n",
               tid, nthreads, getenv("HOSTNAME"));
    }
    return 0;
}
```

```
EOF

# Compilar con soporte OpenMP
gcc -O2 -fopenmp hello_omp.c -o hello_omp

# Ejecutar (una sola tarea, varios hilos)
./hello_omp

echo "==> Fin en: $(date)"
```

4.1.2.2. Trabajos MPI (Múltiple nodos y memoria distribuida)

Ejemplo 3: En el siguiente ejemplo, una aplicación MPI iniciará 8 tareas en 4 nodos, solicitando un límite de tiempo de 1 hora en la partición standard.

```
#!/bin/bash

#SBATCH --job-name=trabajo_paralelo            # Nombre del trabajo
#SBATCH --output=%x_%j.out                     # Archivo de salida
#SBATCH --error=%x_%j.out                      # Archivo de errores
#SBATCH --partition=standard                   # Partición/cola
#SBATCH --nodes=4                              # 4 nodos (pueden ser más)
#SBATCH --ntasks=8                             # 4 tareas/procesos en paralelo
#SBATCH --cpus-per-task=1                      # 1 CPU por tarea
#SBATCH --mem=2G                               # Memoria RAM
#SBATCH --time=00:01:00                        # Tiempo máximo de ejecución
                                                 (horas:minutos:segundos)

# Cargar un módulo (ajustar a disponibilidad del clúster)
module load OpenMPI/4.1.5-GCC-12.3.0

echo "==> Inicio en: $(date)"
echo "==> JobID: ${SLURM_JOB_ID:-N/A}"
echo "==> Directorio de envío: ${SLURM_SUBMIT_DIR:-$PWD}"
echo "==> Nodos asignados: ${SLURM_JOB_NODELIST}"

# Paralelismo sencillo: lanzar N tareas que ejecutan un comando
# (Aquí no usamos un binario MPI; solo demostramos paralelismo)
echo "Hola desde $(hostname) | tarea $SLURM_PROCID de $SLURM_NTASKS"

echo "==> Fin en: $(date)"

# Si tienes un ejecutable MPI (por ejemplo, hello_mpi), sustituye la línea
anterior por:
# mpirun -np $SLURM_NTASKS hello_mpi
```

4.1.2.3. Trabajos Híbridos (Múltiple cores en múltiple nodos)

Ejemplo 4: En este ejemplo se presenta un caso de trabajo híbrido MPI+OMP donde iniciamos un trabajo en 2 nodos, con 4 tareas MPI en cada nodo y 2 hilos OpenMP por tarea MPI. Solicitamos un límite de tiempo de 1 hora en la partición standard.

```
#!/bin/bash

#SBATCH --job-name=trabajo_hibrido              # Nombre del trabajo
#SBATCH --output=%x_%j.out                      # Archivo de salida
#SBATCH --error=%x_%j.out                       # Archivo de errores
#SBATCH --partition=standard                    # Partición/cola
#SBATCH --nodes=2                               # 1 nodo (puedes pedir más)
#SBATCH --ntasks=4                              # 4 rangos MPI (procesos)
#SBATCH --cpus-per-task=2                       # 2 hilos OpenMP por rango
#SBATCH --mem=4G                                # Memoria RAM
#SBATCH --time=00:01:00                         # Tiempo máximo de ejecución
                                                  (horas:minutos:segundos)

# Cargar un módulo (ajustar a disponibilidad del clúster)
module load foss/2023a

echo "==> Inicio en: $(date)"
echo "==> JobID: ${SLURM_JOB_ID:-N/A}"
echo "==> Directorio de envío: ${SLURM_SUBMIT_DIR:-$PWD}"
echo "==> Rangos MPI (SLURM_NTASKS): ${SLURM_NTASKS}"
echo "==> Hilos OpenMP por rango: ${OMP_NUM_THREADS}"

# Hilos OpenMP = CPUs por tarea asignadas por SLURM
export OMP_NUM_THREADS="${SLURM_CPUS_PER_TASK}"

# Programa de ejemplo híbrido (MPI + OpenMP)
cat > hello_hybrid.c <<'EOF'
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#ifdef _OPENMP
 #include <omp.h>
#else
 static int omp_get_thread_num(void){return 0;}
 static int omp_get_num_threads(void){return 1;}
#endif

int main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int nth = omp_get_num_threads();
        #pragma omp critical
        {
            printf("Hola: rank %d/%d, hilo %d/%d en host %s\n",
                   rank, size, tid, nth, getenv("HOSTNAME"));
            fflush(stdout);
        }
    }
```

```
    MPI_Finalize();
    return 0;
}
EOF

# Compilar con soporte híbrido
mpicc -O2 -fopenmp hello_hybrid.c -o hello_hybrid

# Ejecutar: SLURM lanzará 4 rangos MPI, cada uno con 2 hilos OpenMP
./hello_hybrid

echo "==> Fin en: $(date)"
```

4.1.3. Trabajos con GPU

Ejemplo 5: En este script, solicitaremos nodos con GPUs, especificaremos el número de GPUs por nodo y lo **mandamos a la cola/partición "gpu"**.

```
#!/bin/bash

#SBATCH --job-name=trabajo_GPU              # Nombre del trabajo
#SBATCH --output=%x_%j.out                  # Archivo de salida
#SBATCH --error=%x_%j.out                   # Archivo de errores
#SBATCH --partition=gpu                     # Partición/cola
#SBATCH --nodes=1                           # 1 nodo
#SBATCH --ntasks=1                          # 1 proceso
#SBATCH --cpus-per-task=4                   # 4 CPUs de apoyo para el proceso
#SBATCH --mem=8G                            # Memoria RAM
#SBATCH --gres=gpu                          # Solicita recurso de GPU
#SBATCH --time=00:01:00                     # Tiempo máximo de ejecución
                                              (horas:minutos:segundos)

# Cargar un módulo (ajustar a disponibilidad del clúster)
module load CUDA/12.9.0

echo "==> Inicio en: $(date)"
echo "==> JobID: ${SLURM_JOB_ID:-N/A}"
echo "==> Directorio de envío: ${SLURM_SUBMIT_DIR:-$PWD}"
echo "==> Nodo: $(hostname)"

# Comando de prueba: mostrar el estado de la(s) GPU(s)
nvidia-smi

echo "==> Fin en: $(date)"
```

4.1.4. Arrays de trabajo

Slurm soporta Arrays de trabajos y ofrece un mecanismo eficiente para gestionar estas colecciones de trabajos. Los arrays de trabajos son compatibles únicamente con el comando sbatch y pueden definirse utilizando las opciones --array o -a. Para

referirse a un trabajo en un array, Slurm proporciona un ID de array base y un índice de trabajo único para cada trabajo. El formato para especificar un trabajo en un array es primero la ID de trabajo base seguida de un guion bajo "_" y luego el índice del trabajo:

```
<base job id>_<array index>
```

Slurm exporta dos variables de entorno que pueden utilizarse en el script de trabajo para identificar cada trabajo del array:

- SLURM_ARRAY_JOB_ID: ID de trabajo base del array.
- SLURM_ARRAY_TASK_ID: Índice del trabajo.

Además, existen opciones adicionales para especificar los nombres de los archivos de entrada estándar, salida estándar y error estándar. La opción %A se reemplazará por el valor de SLURM_ARRAY_JOB_ID y la opción %a se reemplazará por el valor de SLURM_ARRAY_TASK_ID.

Cada trabajo en un array tiene su propia ID de trabajo única. Esta ID se exporta en la variable de entorno **SLURM_JOBID.**

Ejemplo 6: En el siguiente ejemplo, el script de trabajo creará un array de 10 trabajos con índices del 0 al 9. Cada trabajo se ejecutará en 1 nodo con un límite de tiempo de 1 hora.

```
#!/bin/bash

#SBATCH --job-name=array_trabajos         # Nombre del trabajo
#SBATCH --output=array_trabajos-%A_%a.out # Archivo de salida (%A para el ID
del trabajo principal, %a para el ID del array)
#SBATCH --error=array_trabajos-%A_%a.err  # Archivo de errores (%A para el ID
del trabajo principal, %a para el ID del array)
#SBATCH --partition=standard              # Partición donde ejecutar el trabajo
#SBATCH --nodes=1                         # Número de nodos por tarea
#SBATCH --ntasks=1                        # Número de tareas por trabajo
#SBATCH --cpus-per-task=1                 # Número de CPUs por tarea
#SBATCH --mem=2G                          # Memoria por nodo
#SBATCH --time=01:00:00                   # Tiempo máximo de ejecución
                                           (horas:minutos:segundos)
#SBATCH --array=0-9                       # Rango del array de trabajos (aquí
                                            se ejecutan 10 trabajos, de 0 a 9)

# Usar el ID del array para determinar la entrada específica para este trabajo
input_file="input_${SLURM_ARRAY_TASK_ID}.txt"

# Ejecutar el programa con la entrada específica
```

```
./mi_programa ${input_file}
```

5. Trabajos interactivos

Este tipo de trabajos están concebidos exclusivamente para la realización de **pruebas breves** dentro del entorno del clúster. Para ejecutar trabajos finales, de mayor duración o que deban ejecutarse de forma desatendida, es imprescindible enviarlos al **gestor de colas del clúster mediante un script de trabajo (job script)**.

Las sesiones interactivas se pueden asignar utilizando el comando `salloc`. El siguiente comando, por ejemplo, asignará 16 GB de memoria RAM y 4 núcleos de CPU durante 6 horas:

```
salloc --mem=16G -c 4 -t 06:00:00 -p standard srun --pty /bin/bash -i
```

Una vez realizada la asignación, el comando salloc iniciará un bash en el nodo de inicio de sesión donde se realizó la solicitud. Tras una asignación exitosa, los usuarios pueden ejecutar interactivamente sus aplicaciones. Por ejemplo:

```
$ module load R/4.2.0-foss-2021b
$ R CMD BATCH --no-save simulacion.R simulacion.Rout
```

Nota: Las sesiones interactivas tienen un límite de tiempo de 24 horas.

6. Flujo de trabajo

A continuación se muestran ejemplos de cómo un usuario puede interactuar con el clúster para mandar trabajos:

6.1. Lanzar un trabajo al clúster

Este ejemplo describe de manera genérica los pasos que un usuario debe seguir para enviar un trabajo por lotes al clúster.

1. **Conectar con la VPN del centro:** El usuario deberá instalar y configurar correctamente la VPN del centro. Las instrucciones necesarias habrán sido proporcionadas al usuario en el momento de su registro a través del formulario del centro.

   Más información en el punto 1.4. Acceso al clúster

2. **Conectar con los nodos de acceso al clúster:** Una vez verificada la conexión

correcta a la VPN, el siguiente paso es ingresar al clúster de supercomputación utilizando SSH con su usuario y contraseña. Usando un terminal o un cliente SSH, el usuario debe conectarse a *login.spc.cica.es*:

```
$ ssh usuario@login.spc.cica.es
```

3. **Subir datos necesarios para la ejecución del trabajo:** Teniendo acceso al clúster y a su espacio de trabajo ($HOME), el usuario debe subir los archivos de entrada o datos necesarios para ejecutar su trabajo. Para más información sobre el límite de espacio disponible para cada usuario, consulte el punto 1.3. Gestión de datos – Sistema de Ficheros.
4. **Realizar pruebas con el software a ejecutar:** El siguiente paso es asegurarse de cómo se ejecuta el programa dentro del clúster, cómo invocar el programa, qué parámetros recibirá, qué recursos (CPU, RAM) necesitará, etc. Para ello, los usuarios disponen de sesiones interactivas en las que pueden trabajar con un terminal dentro del clúster para realizar pruebas o compilar sus propios programas.

   Dado que la finalidad de este tipo de sesiones es la realización de pruebas y no la ejecución de trabajos de computación, el límite máximo de solicitud es de 24 horas.

   Para iniciar una sesión interactiva, se debe usar el siguiente comando:

   ```
   salloc --mem=16G -c 4 -t 06:00:00 -p standard srun --pty /bin/bash -i
   ```

   Para más información sobre este tipo de sesiones/trabajos, consultar el punto 5. Trabajos interactivos.

   - **Cargar módulos**: Cargue los módulos necesarios para su trabajo. Por ejemplo, para cargar un programa específico, use:

   ```
   $ module load FreeFem++/4.11
   ```

   Más información en el punto 1.5. Módulos

   - **Ejecutar el programa:** Lance el programa con el que desea trabajar:

   ```
   $ FreeFem++ problem.edp
   ```

5. **Escribir el script de trabajo:** Cuando tenga claro cómo se ejecuta su programa en el clúster, debe escribir un script de trabajo para enviarlo al gestor de colas. Para más información sobre cómo redactar estos scripts y los parámetros que deben contener, consulte el punto 4.1. Envío de trabajos por

lotes.

6. **Enviar el trabajo a la cola:** Envíe su trabajo a la cola del gestor de recursos:

```
$ sbatch <script_de_trabajo>
```

7. **Monitoriza el trabajo:** Verifique el estado de su trabajo usando:

```
$ squeue -u <usuario>
```

   Más información en el punto 3.3. Monitorización de los trabajos

8. **Revisa los resultados:** Una vez completado el trabajo, revise los archivos de salida y error (mi_trabajo.out y mi_trabajo.err) para verificar que todo se ejecutó correctamente.

7. Soporte

Para cualquier duda o consulta relacionada con el uso del clúster, por favor envíe un correo a la dirección soporte@cica.es