Pluton User's Guide

Versión 2.4.2

July 31, 2023

PROYECTO UNLC10-1E728

Autor:

Roberto R. Expósito

Contents

1 Introducción **3**

2 Descripción General del Clúster **3**
2.1 Nodo frontend . . . 4
2.2 Nodos compute-X-Y . . . 4

3 Conexión al Clúster **5**
3.1 Cambio de password . . . 7
3.2 Transferencia de ficheros . . . 7

4 Sistemas de Ficheros **8**

5 Redes de Interconexión **9**

6 Ejecución de Trabajos **9**
6.1 Slurm Workload Manager . . . 10
6.2 Parámetros obligatorios . . . 10
6.3 Trabajos batch secuenciales . . . 10
6.3.1 Solicitando cores . . . 12
6.3.2 Solicitando memoria . . . 12
6.3.3 Solicitando acceso exclusivo . . . 13
6.3.4 SMT . . . 13
6.4 Recursos especiales (GPUs) . . . 13
6.5 Particiones . . . 15
6.6 Trabajos batch paralelos . . . 17
6.6.1 Aplicaciones OpenMP/Threads . . . 17
6.6.2 Aplicaciones MPI . . . 18
6.6.3 Aplicaciones híbridas MPI+OpenMP . . . 20
6.6.4 Aplicaciones híbridas MPI+CUDA/OpenCL . . . 21
6.7 Almacenamiento temporal . . . 22
6.7.1 Local . . . 22
6.7.2 Global . . . 23
6.8 Trabajos interactivos . . . 25
6.8.1 X11 forwarding . . . 26
6.9 Acceso por ssh a los nodos de cómputo . . . 28
6.10 Ejecución de contenedores Docker . . . 29
6.10.1 Ejemplos de uso . . . 30
6.10.2 GPUs NVIDIA en contenedores . . . 32
6.11 Avisos por correo electrónico . . . 34
6.12 Otros comandos útiles . . . 34

7 Entorno de Software **35**
7.1 Lmod . . . 35
7.1.1 Lmod vs Enviroment Modules . . . 39

8 Ayuda **40**

1 Introducción

La guía de usuario del clúster Pluton [1] pretende proporcionar la información mínima necesaria para un usuario nuevo del sistema. Por tanto, se asume que el usuario es familiar con las utilidades más comunes de las plataformas GNU/Linux. La última versión de la guía puede ser descargada por los usuarios del clúster desde [2].

Lea detenidamente el siguiente documento y no dude en contactar con el administrador (ver Ayuda) en caso de alguna duda o problema.

2 Descripción General del Clúster

Pluton es un clúster heterogéneo para computación de altas prestaciones (*High Performance Computing*, HPC) que ha sido cofinanciado por el Ministerio de Economía y Competitividad y por el Fondo Europeo de Desarrollo Regional (FEDER), inicialmente a través del proyecto **UNLC10-1E-728**, y que ido recibiendo pequeñas actualizaciones sucesivas a través de nuevo proyectos.

El clúster se compone de un único punto de entrada accesible desde el exterior (nodo de login o nodo *frontend*) denominado **pluton.dec.udc.es**. A este nodo es donde los usuarios se conectan al clúster para editar/compilar sus códigos y enviar trabajos al planificador o sistemas de colas para su posterior ejecución en los nodos de cómputo. Por tanto, **NO está permitido la ejecución de trabajos en el nodo *frontend***. Los nodos de cómputo son los encargados de proporcionar los recursos computacionales del clúster tales como CPU, memoria y aceleradoras (e.g., GPU). Estos nodos de cómputo se agrupan de forma lógica en cabinas de cómputo, donde cada nodo tiene asociado un número de cabina (X) y un número de nodo dentro de la cabina (Y), de forma que son nombrados como **compute-X-Y**. Por ejemplo, **compute-0-0** se refiere al primer nodo de cómputo de la primera cabina.

El nodo *frontend* también hace la función de servidor Network Attached Storage (**NAS**) en el clúster. En el servidor NAS se almacenan todos los ficheros de los usuarios, es decir, es donde residen físicamente los directorios $HOME, que es lo que se considera como almacenamiento fijo o permanente. Dichos ficheros son accedidos remotamente desde los nodos de cómputo a través de la red mediante NFS. La Figura 1 muestra la estructura general del clúster Pluton.

Cada de una de las cabinas agrupa un número variable de nodos de cómputo que suelen compartir características hardware muy similares. En la actualidad, Pluton consta de cuatro cabinas de cómputo:

- **Cabina 0**: 15 nodos de cómputo (**compute-0-[0-3] y compute-0-[6-16]**) para un total de 240 cores físicos (480 threads), 960 GiB de memoria y 15 aceleradoras NVIDIA Tesla GPU (Kepler).
- **Cabina 1**: 2 nodos de cómputo (**compute-1-[0-1]**) para un total de 48 cores físicos (96 threads) y 256 GiB de memoria.
- **Cabina 2**: 10 nodos de cómputo (**compute-2-[0-9]**) con un total de 320 cores físicos (640 threads), 2560 GiB de memoria y 2 aceleradoras NVIDIA Tesla GPU (Turing).
- **Cabina 3**: 1 nodo de cómputo (**compute-3-0**) para un total de 48 cores físicos (96 threads) y 256 GiB de memoria.

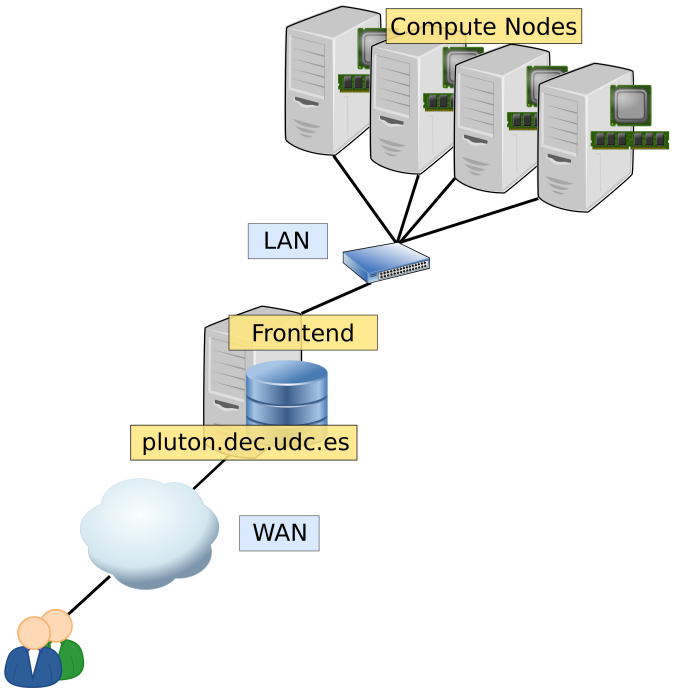

Figure 1: Clúster Pluton

- **Cabina 4**: 2 nodo de cómputo (**compute-4-[0-1]**) para un total de 64 cores físicos (128 threads) y 512 GiB de memoria.

En resumen, el clúster Pluton dispone de: **30 nodos de cómputo** para un total de **720 cores físicos (1440 threads)**, **4.4 TiB de memoria** y **17 aceleradoras NVIDIA Tesla GPU**.

A continuación se detallan las principales características hardware del nodo *frontend* y de los nodos de cómputo que componen cada cabina.

2.1 Nodo frontend

La Tabla 1 muestra las características hardware del nodo de login o *frontend* del clúster.

2.2 Nodos compute-X-Y

Las Tablas 2, 3, 4, 5 y 6 muestran las características hardware más relevantes de los nodos de cómputo de las diferentes cabinas.

	pluton.dec.udc.es
CPU Model	Intel Xeon Silver 4110 Skylake-SP
CPU Speed/Turbo	2.10 GHz/3.0 GHz
#Cores por CPU	8
#Threads por core	2
#Cores/Threads por nodo	8/16
Cache L1/L2/L3	32 KiB/1 MiB/11 MiB
Memoria RAM	128 GiB DDR4 2666 Mhz (4 × 32 GiB)
Discos	2 × SDD 240 GiB SATA3 5 × HDD 4 TiB SATA3 7.2K rpm 2 × HDD 2 TiB SATA3 7.2K rpm 5 × HDD 1 TiB SATA3 7.2K rpm
Controladora RAID	HPE Smart Array P816i-a SR Gen10 4 GiB
Redes	InfiniBand EDR & Gigabit Ethernet

Tabla 1: Descripción del nodo *frontend* del clúster

	compute-0-{0-3}, compute-0-{6-16}
CPU Model	2 × Intel Xeon E5-2660 Sandy Bridge-EP
CPU Speed/Turbo	2.20 GHz/3.0 GHz
#Cores por CPU	8
#Threads por core	2
#Cores/Threads por nodo	16/32
Cache L1/L2/L3	32 KiB/256 KiB/20 MiB
Memoria RAM	64 GiB DDR3 1600 Mhz (8 × 8 GiB)
Discos	1 × HDD 1 TiB SATA3 7.2K rpm
Aceleradoras	1 × NVIDIA Tesla Kepler K20m 5 GiB GDDR5 (0-{0-12}) 1 × NVIDIA Tesla Kepler K40c 12 GiB GDDR5 (0-13) 3 × NVIDIA Tesla Kepler K20m 5 GiB GDDR5 (0-15)
Redes	InfiniBand FDR & Gigabit Ethernet

Tabla 2: Descripción de los nodos de cómputo de la cabina 0

	compute-1-{0-1}
CPU Model	2 × Intel Xeon E5-2650v4 Broadwell-EP
CPU Speed/Turbo	2.20 GHz/2.90 GHz
#Cores por CPU	12
#Threads por core	2
#Cores/Threads por nodo	24/48
Cache L1/L2/L3	32 KiB/256 KiB/30 MiB
Memoria RAM	128 GiB DDR4 2400 Mhz (4 × 32 GiB)
Discos	1 × HDD 1 TiB SATA3 7.2K rpm
Redes	InfiniBand FDR & Gigabit Ethernet

Tabla 3: Descripción de los nodos de cómputo de la cabina 1

3 Conexión al Clúster

La conexión al clúster Pluton se realiza a través del nodo de login o *frontend*, cuyo DNS de acceso es: **pluton.dec.udc.es**. Una vez que recibes del administrador tu nombre de usuario (*username*) y contraseña (*password*) el acceso se realiza a través de un canal

	compute-2-{0-9}
CPU Model	2 × Intel Xeon Silver 4216 Cascade Lake-SP
CPU Speed/Turbo	2.1 GHz/3.2 GHz
#Cores por CPU	16
#Threads por core	2
#Cores/Threads por nodo	32/64
Cache L1/L2/L3	32 KiB/1 MiB/22 MiB
Memoria RAM	256 GiB DDR4 2933 Mhz (8 × 32 GiB)
Discos	1 × HDD 2 TiB SATA3 7.2K rpm 1 × SDD 240 GiB SATA3
Aceleradoras	2 × NVIDIA Tesla T4 16 GiB GDDR6 (2-0)
Redes	InfiniBand EDR & Gigabit Ethernet

Tabla 4: Descripción de los nodos de cómputo de la cabina 2

	compute-3-0
CPU	2 × Intel Xeon Gold 5220R Cascade Lake-SP Refresh
CPU Speed/Turbo	2.2 GHz/4.0 GHz
#Cores por CPU	24
#Threads por core	2
#Cores/Threads por nodo	48/96
Cache L1/L2/L3	32 KiB/1 MiB/35.75 MiB
Memoria RAM	256 GiB DDR4 2933 Mhz (8 × 32 GiB)
Discos	2 × SDD 960 GiB SATA3
Redes	InfiniBand EDR & Gigabit Ethernet

Tabla 5: Descripción de la cabina 3

	compute-4-{0-1}
CPU Model	2 × Intel Xeon Silver 5218 Cascade Lake-SP
CPU Speed/Turbo	2.3 GHz/3.9 GHz
#Cores por CPU	16
#Threads por core	2
#Cores/Threads por nodo	32/64
Cache L1/L2/L3	32 KiB/1 MiB/22 MiB
Memoria RAM	256 GiB DDR4 2933 Mhz (8 × 32 GiB)
Discos	1 × HDD 2 TiB SATA3 7.2K rpm 1 × SDD 240 GiB SATA3
Redes	InfiniBand EDR & Gigabit Ethernet

Tabla 6: Descripción de los nodos de cómputo de la cabina 4

seguro utilizando un cliente **ssh**. Un ejemplo de conexión desde una sistema operativo GNU/Linux sería:

```
[rober@oceania ~]$ ssh -l username pluton.dec.udc.es
```

3.1 Cambio de password

Para cambiar la password de acceso al clúster, basta con invocar el comando estándar en sistemas GNU/Linux: **passwd**. Será **obligatorio** cambiar la password cuando se reciben los datos de acceso para una nueva cuenta de usuario en el clúster, pues se entrega por defecto una password muy débil que sólo debe ser utilizada para la primera conexión.

A continuación se muestra un ejemplo para cambiar la password de usuario:

```
-bash-3.2$ passwd
Changing password for user username.
Changing password for username
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

3.2 Transferencia de ficheros

Para la transferencia de ficheros desde/hacia el clúster se pueden utilizar herramientas seguras como **scp** o **sftp**. A continuación se muestran algunos ejemplos de uso básicos de estas herramientas.

Ejemplos de transferencia de un fichero desde tu máquina local al clúster:

```
[rober@oceania ~]$ scp localfile username@pluton.dec.udc.es:
username@pluton.dec.udc.es's password:
localfile                     100%   25MB  12.4MB/s   00:02
[rober@oceania ~]$
```

```
[rober@oceania ~]$ sftp username@pluton.dec.udc.es
Connecting to pluton.dec.udc.es...
username's password:
sftp> put localfile
Uploading localfile to /home/username/localfile
localfile                     100%   25MB   8.2MB/s   00:03
sftp> bye
[rober@oceania ~]$
```

Ejemplos de transferencia de un fichero desde el clúster a tu máquina local:

```
[rober@oceania ~]$ scp username@pluton.dec.udc.es:remotefile .
username@pluton.dec.udc.es's password:
localfile                     100%   25MB   1.6MB/s   00:16
[rober@oceania ~]$
```

```
[rober@oceania ~]$ sftp username@pluton.dec.udc.es
Connecting to pluton.dec.udc.es...
username's password:
sftp> get remotefile
```

```
Fetching /home/username/remotefile to remotefile
localfile                          100%   25MB   1.6MB/s   00:16
sftp> bye
[rober@oceania ~]$
```

4 Sistemas de Ficheros

Cada nodo del clúster Pluton cuenta con diferentes sistemas de ficheros, cada uno con un tamaño y características diferentes. Estos son lo más relevantes:

1. **Root (/)**: es donde reside el sistema operativo de cada nodo, y sólo puede contener datos del mismo, por tanto un usuario no tendrá permisos para almacenar/modificar ficheros en las rutas críticas del sistema. Este sistema de ficheros incluye una ruta para archivos temporales (/**tmp**), pero debido a su pequeño tamaño no debe ser usado **bajo ningún concepto** para almacenamiento temporal. Para dicho propósito se debe usar el siguiente sistema de ficheros.

2. **Scratch (/scratch)**: Cada nodo de cómputo dispone de un sistema de ficheros en el disco local que puede ser usado por los trabajos en ejecución para **almacenamiento temporal en local** (también llamado almacenamiento en *scratch*). Este sistema de ficheros está montado en los nodos de cómputo bajo la ruta /**scratch**, y su tamaño varía dependiendo del nodo de cómputo en concreto (e.g., los nodos compute-0-x disponen de unos 850 GiB). Todos los datos almacenados en esta ruta local de los nodos de cómputo no estarán disponibles desde el nodo *frontend*. Además, estos datos no son eliminados automáticamente al acabar la ejecución de las aplicaciones, por tanto sería responsabilidad del usuario eliminarlos (o no) al finalizar en función de sus necesidades.

3. **Scratch SSD (/scratch/ssd)**: Algunos nodos de cómputo disponen de un sistema de ficheros adicional en un disco local de tipo **SSD** para **almacenamiento temporal**. Este sistema de ficheros es accesible en la ruta /**scratch**/**ssd** para aquellos nodos que disponen del mismo. Al igual que el caso anterior, los datos almacenados en esta ruta local de los nodos de cómputo no estarán disponibles desde el nodo *frontend*. Además, estos datos tampoco son eliminados automáticamente, por tanto cada usuario debe gestionarlos según sus necesidades.

4. **$HOME (/home)**: Como se ha dicho anteriormente, el nodo *frontend* realiza a su vez las funciones de un dispositivo NAS, y es donde residen físicamente los directorios **$HOME** de los usuarios para proporcionar un **almacenamiento global persistente** de sus ficheros. Estos directorios son montados mediante NFS en todos los nodos de cómputo haciendo uso de la red InfiniBand (ver Redes de Interconexión). Por tanto, la NAS se encarga de exportar los directorios $HOME a los nodos de cómputo, los cuales están almacenados en configuración RAID 10 (por hardware) usando 4 discos, proporcionando así una alta tolerancia a fallos. Además, se realiza semanalmente un *backup* de los directorios $HOME, el cual es almacenado en un sistema de ficheros independiente también en configuración RAID 1, para aumentar la redundancia y seguridad de los datos. Este sistema de ficheros **tiene activado**

el sistema de cuotas para limitar la cantidad de datos que los usuarios pueden almacenar. Para ver la cuota máxima asignada, así como el espacio en disco utilizado hasta el momento actual, utilice el comando **quota**.

5. **$STORE (/share/store)**: Este sistema de ficheros constituye un segundo espacio de **almacenamiento global persistente** a disposición de los usuarios, accesible vía **$STORE**, que también es montado mediante NFS en todos los nodos de cómputo usando la red InfiniBand. Los datos en $STORE están almacenados físicamente en configuración RAID 10 usando 4 discos. Este sistema de ficheros **no dispone de sistema de cuotas**, lo que permite a los usuarios manejar mayores volúmenes de datos que en $HOME.

IMPORTANTE: Es responsabilidad del usuario hacer un *backup* de sus datos más críticos, pues como ya se indicó anteriormente sólo se garantiza un *backup* semanal de su directorio $HOME.

IMPORTANTE: No se hace ningún *backup* de los datos almacenados en $STORE.

5 Redes de Interconexión

La Figura 1 muestra la estructura general del clúster Pluton, donde se ve que todos los nodos están interconectados mediante una red LAN. Esta red de interconexión es una red de tipo **Gigabit Ethernet**, con un ancho de banda máximo de **1 Gbps**, y que está presente en todos los nodos del clúster.

Adicionalmente a esta red Gigabit Ethernet, tal y como se puede ver en la descripción de los nodos, estos cuentan con una red de interconexión de altas prestaciones (baja latencia y alto ancho de banda): **InfiniBand (IB)** [5]. En IB, existen diferentes especificaciones que determinan el máximo ancho de banda que permite la red, actualmente y de menor a mayor ancho de banda existen la siguientes especificaciones: SDR, DDR, QDR, FDR10, FDR, EDR y HDR. Todos los nodos del clúster cuentan con una interfaz de red **FDR o EDR** contectados a un switch FDR, permitiendo anchos de banda de hasta **56 Gbps** y latencias en torno a **1-2** μ**s**. Esta red es la más adecuada para aquellas aplicaciones paralelas MPI muy intensivas en comunicaciones, ya que permitirá una mejor escalabilidad que utilizando Gigabit Ethernet. Además, como se ha dicho anteriormente, el nodo *frontend* sirve los directorios $HOME y $STORE de los usuarios mediante NFS bajo la red InfiniBand.

6 Ejecución de Trabajos

La ejecución de trabajos se realizará **exclusivamente en los nodos de cómputo** utilizando para ello un **planificador/gestor de trabajos/sistema de colas**. Este software se encarga de distribuir los trabajos de los usuarios y obtener los recursos necesarios para los mismos. Por tanto, los usuarios enviarán sus trabajos al sistema de colas desde el nodo *frontend*, y éstos serán ejecutados en los nodos de cómputo cuando el planificador lo considere oportuno (dependiendo de la carga del sistema y número de usuarios, número de trabajos totales, políticas de uso, etc).

6.1 Slurm Workload Manager

En Pluton se encuentra instalado el gestor de trabajos denominado **Slurm Workload Manager** [6] en su versión 19.05.2. Un **trabajo** (*job*) es la unidad de ejecución en Slurm, el cuál se envía al gestor para su ejecución utilizando la línea de comandos. Existen principalmente dos tipos de trabajos: ***batch*** e **interactivo**. A continuación, se muestra de forma breve como ejecutar trabajos sencillos utilizando Slurm explicando únicamente las opciones más básicas. Para una referencia completa, puede consultar las páginas del manual de cada comando o la documentación de Slurm disponible en [7].

6.2 Parámetros obligatorios

Todos los usuarios deben especificar de forma **obligatoria** una estimación de la duración máxima del trabajo mediante el parámetro **-t|--time**, cuyo valor se expresa en formato Horas:Minutos:Segundos (e.g., -t=00:10:00) o directamente en minutos (e.g., -t=10). Si se excede la duración especificada nuestro trabajo recibirá primero la señal SIGTERM, la cuál puede ser capturada para finalizar la aplicación de forma normal, y finalmente será terminado forzosamente pasados 30 segundos mediante la señal SIGKILL.

Especificar la duración de los trabajos permite a Slurm mejorar la planificación de los mismos y aprovechar de forma más eficiente los recursos, ya que puede priorizar trabajos cortos y que necesitan pocos recursos para planificarlos antes que otros trabajos de mayor duración que están a la espera de que se liberen algunos de los recursos que necesitan (técnica *backfilling*). Esto implica que **cuanto mejor sea la estimación de la duración de nuestro trabajo, más posibilidades tendremos de que se ejecuten antes**.

IMPORTANTE: Todos los trabajos, tanto *batch* como interactivos, están limitados a una duración máxima de **72 horas** (-t=72:00:00), con el fin de que los usuarios no puedan monopolizar los nodos de cómputo durante largos períodos de tiempo.

IMPORTANTE: El número máximo de trabajos que un usuario puede tener en ejecución de forma simultánea es de **5**, y en total puede tener hasta **25** trabajos activos, que incluye tanto trabajos en ejecución (estado R) como trabajos en espera (estado PD).

6.3 Trabajos batch secuenciales

Un trabajo *batch* es una secuencia de comandos especificados en un script. Un ejemplo de trabajo *batch* **secuencial** podría ser:

```
#!/bin/bash
#
# This is a simple example of a slurm batch script
#
# print date and time
date
# Sleep for 20 seconds
sleep 20
# print date and time again
date
```

Si se asume que el anterior script es guardado con el nombre de simple.sh, para enviarlo a ejecución se utilizaría el comando **sbatch**, especificando de forma obligatoria la duración máxima estimada del trabajo con -t|--time. Un ejemplo sería:

```
[root@pluton ~]# sbatch -t 00:10:00 simple.sh
Submitted batch job 109
```

Para obtener información sobre el estado del trabajo enviado anteriormente se utiliza el comando **squeue**. Un ejemplo de la salida de este comando sería:

```
[root@pluton ~]# squeue
  JOBID PARTITION  NAME     USER  ST  TIME NODES  NODELIST(REASON)
    109 compute0   simple.s rober  R  0:01     1  compute-0-0
```

Este comando nos indicará, entre otras cosas, el estado actual de nuestros trabajos en la quinta columna, donde por ejemplo "R" significa que está en ejecución o "PD" que está en espera (pendiente) para ser planificado y ejecutado.

Una vez ejecutado nuestro trabajo, su salida la obtendremos en un fichero nombrado como "slurm-%j.out", siendo %j el identificador del trabajo (**$SLURM_JOB_ID**) devuelto por el comando sbatch. Para el ejemplo anterior este fichero de salida es **slurm-109.out**, cuyo contenido se muestra a continuación:

```
[root@pluton ~]# cat slurm-109.out
mié oct  9 09:44:54 CEST 2019
mié oct  9 09:45:14 CEST 2019
```

Un parámetro útil de sbatch es **-J|--job-name**, el cuál nos permite cambiar el nombre de un trabajo (por defecto el nombre de un trabajo es igual al nombre del script). Por tanto, si ejecutamos el anterior script de esta forma:

```
[root@pluton ~]# sbatch -J test -t 00:10:00 simple.sh
Submitted batch job 110
[root@pluton ~]# squeue
  JOBID PARTITION  NAME     USER  ST  TIME NODES  NODELIST(REASON)
    110 compute0   test     rober  R  0:01     1  compute-0-0
```

Se observa como **squeue** muestra el nombre especificado con -J, pero la salida del trabajo la obtendríamos en este caso en el fichero **slurm-110.out**. Para cambiar el nombre del fichero de salida se puede usar el parámetro **-o|--output**, que además nos permite obtener el nombre e identificador del trabajo mediante los símbolos %x y %j, respectivamente. Ejemplo de uso:

```
[root@pluton ~]# sbatch -J test -o %x_%j.out -t 00:10:00 simple.sh
Submitted batch job 112
[root@pluton ~]# cat test_112.out
mié oct  9 09:46:51 CEST 2019
mié oct  9 09:47:11 CEST 2019
```

Se puede observar que ahora el fichero de salida es **test_112.out**. Estos parámetros de sbatch (-J y -o) al igual que muchos otros (incluído -t), también pueden ser especificados en el propio script en vez de por línea de comandos. Para ello, la directiva **#SBATCH** es utilizada dentro del script para especificar parámetros. Por ejemplo, si queremos especificar -J y -o en el propio script, utilizaríamos:

```
#!/bin/bash
#
#SBATCH -J test
#SBATCH -o %x_%j.out
#
date
sleep 20
date
```

6.3.1 Solicitando cores

Los parámetros más importantes para especificar el número cores que necesitamos son:

- **-n, --ntasks=<number>**: número máximo de tareas (procesos) que vamos a ejecutar. Por defecto: -n 1.
- **-c, --cpus-per-task=<ncpus>**: número máximo de cores por tarea (por proceso) que vamos a usar. Por defecto: -c 1.

Estos valores se pueden obtener desde el entorno de nuestro trabajo ya que Slurm define variables apropiadas: **$SLURM_NTASKS** y **$SLURM_CPUS_PER_TASK**. A continuación se muestra un ejemplo que muestra el valor de ambas variables cuando se solicitan 2 tareas y 4 cores por tarea:

```
#!/bin/bash
#
#SBATCH -J test
#SBATCH -o %x_%j.out
#
echo SLURM_NTASKS=$SLURM_NTASKS
echo SLURM_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK
[root@pluton ~]# sbatch -n 2 -c 4 -t 00:10:00 simple.sh
Submitted batch job 182
[root@pluton ~]# cat test_182.out
SLURM_NTASKS=2
SLURM_CPUS_PER_TASK=4
```

En los ejemplos de la Sección 6.3 no estábamos especificando ninguno de estos dos parámetros. Por tanto se estaba solicitando ejecutar una única tarea usando un core en un nodo de cómputo, válido para la ejecución de trabajos secuenciales. El uso de los parámetros -n y -c está principalmente orientado a la ejecución de aplicaciones paralelas (ver Trabajos batch paralelos).

6.3.2 Solicitando memoria

Los requisitos de memoria de nuestros trabajos se pueden especificar mediante dos parámetros que son mutuamente exclusivos:

- **--mem=<size[units]>**: permite especificar la memoria que requiere nuestro trabajo por nodo de cómputo. La unidad por defecto son megabytes, aunque se pueden especificar otras unidades: [K|M|G|T].

- **--mem-per-cpu=<size[units]>**: permite especificar la memoria que requiere nuestro trabajo por cada core solicitado. La unidad por defecto son megabytes, aunque se pueden especificar otras unidades: [K|M|G|T].

Ejemplos de uso:

```
[root@pluton ~]# sbatch --mem=8192 -t 00:10:00 simple.sh
Submitted batch job 194
[root@pluton ~]# sbatch -c 2 --mem-per-cpu=2G -t 00:10:00 simple.sh
Submitted batch job 195
```

En el primer ejemplo estamos solicitando un nodo de cómputo que tenga 8192 MiB (8 GiB) de memoria disponible. En el segundo caso solicitamos 2 cores con 2 GiB por core (4 GiB en total). Al igual que antes, también es posible obtener los valores desde el entorno del trabajo mediante las variables **$SLURM_MEM_PER_NODE** y **$SLURM_MEM_PER_CPU**.

6.3.3 Solicitando acceso exclusivo

La opción **--exclusive** permite solicitar los nodos de cómputo de forma exclusiva: ningún otro trabajo de ningún otro usuario será planificado en dichos nodos hasta que nuestro trabajo termine (siempre y cuando los recursos solicitados estuvieran disponibles). Además, usando esta opción siempre se nos asignan todos los cores y la memoria disponible en los nodos. Este parámetro es especialmente útil cuando estamos haciendo *benchmarking* o mediciones del tiempo de ejecución de una aplicación y necesitamos que otros trabajos no interferieran en nuestras medidas. Ejemplo de uso:

```
[root@pluton ~]# sbatch --exclusive -n 2 -c 2 -t 00:10:00 simple.sh
Submitted batch job 196
```

6.3.4 SMT

Todos los nodos del clúster soportan y tienen activada la tecnología Simultaneous Multithreading (SMT), denominada Hyper-Threading en los procesadores Intel. Sin embargo, por defecto Slurm está configurado para asignar cores "completos" a las tareas. Para poder ejecutar tantas tareas/procesos como cores lógicos en un nodo de cómputo usando SMT hay que especificar el parámetro **-O, --overcommit** junto con **--exclusive**.

6.4 Recursos especiales (GPUs)

Cuando se envía un trabajo a ejecución se pueden especificar parámetros adicionales para solicitar determinados recursos especiales que están disponibles en los nodos de cómputo. Como se ha visto anteriormente, los nodos de cómputo disponen de características hardware diferentes, como modelos de CPUs con distinto número de cores y frecuencia, una mayor o menor cantidad de memoria RAM disponible, diferentes tipos de discos o la disponibilidad de tarjetas aceleradoras como las GPUs.

Para solicitar un tipo de GPU que queremos que tengan los nodos de cómputo dónde se planifiquen nuestros trabajos para su ejecución se debe usar el siguiente parámetro:

- **--gres=gpu[[:type]:count]**: permite especificar que nuestro trabajo requiere un nodo que disponga de una GPU, y opcionalmente podemos especificar el número de GPUs (count) y su modelo (type). Modelos de GPU NVIDIA disponibles: **K20, K40, T4**.

  Slurm define la variable **$CUDA_VISIBLE_DEVICES** para las GPUS NVIDIA especificando las GPUs que han sido asignadas a nuestro trabajo. Si esta variable es modificada en nuestro script es probable que obtengamos errores en el acceso a las GPUs desde nuestra aplicación CUDA.

A continuación se muestra un ejemplo solicitando una GPU de tipo NVIDIA K20:

```
[root@pluton ~]# cat cuda.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out

echo CUDA_VISIBLE_DEVICES=$CUDA_VISIBLE_DEVICES
/share/apps/utils/simpleMultiGPU
[root@pluton ~]# sbatch -t 00:10:00 --gres=gpu:K20:1 cuda.sh
Submitted batch job 129
[root@pluton ~]# cat test_129.out
CUDA_VISIBLE_DEVICES=0
Starting simpleMultiGPU
CUDA-capable device count: 1
Generating input data...

Computing with 1 GPUs...
GPU Processing time: 24.743000 (ms)

Computing with Host CPU...

Comparing GPU and Host CPU results...
GPU sum: 16777296.000000
CPU sum: 16777294.395033
Relative difference: 9.566307E-08
```

Hay que tener especial precaución de no especificar una combinación de recursos que sea imposible de satisfacer por parte del planificador. Si por ejemplo especificamos el parámetro **--gres=gpu:K40:2**, nuestro trabajo no podría ser planificado nunca, pues no existe ningún nodo de cómputo que tenga 2 GPUs del modelo NVIDIA K40. Cuando ésto ocurre, obtendremos el siguiente error:

```
[root@pluton ~]# sbatch --gres=gpu:K40:2 -t 00:10:00 simple.sh
sbatch: error: Batch job submission failed: Requested node
   configuration is not available
```

6.5 Particiones

Una **partición** en Slurm es un concepto muy similar al concepto de **cola** que usan otros gestores de trabajos. Las particiones permiten agrupar los nodos de cómputo en conjuntos (no necesariamente disjuntos). En Pluton se disponen de 5 particiones denominadas **compute0**, **compute1**, **compute2**, **compute3** y **compute4**, una por cada cabina de cómputo (por tanto, disjuntas en este caso). De esta forma, las particiones agrupan los nodos de cada cabina ya que estos suelen disponer de características hardware muy similares o idénticas.

Para ver las particiones disponibles y sus características se puede usar el comando: **scontrol show partition**. Un ejemplo de la ejecución de dicho comando sería:

```
PartitionName=compute0
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=normal
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0
   Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO
   MaxCPUsPerNode=UNLIMITED
Nodes=compute-0-[0-3],compute-0-[6-16]
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO
   OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=240 TotalNodes=15 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerCPU=3800 MaxMemPerNode=60800

PartitionName=compute1
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=normal
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0
   Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO
   MaxCPUsPerNode=UNLIMITED
Nodes=compute-1-[0-1]
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO
   OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=48 TotalNodes=2 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerCPU=5200 MaxMemPerNode=124800

PartitionName=compute2
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=YES QoS=normal
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0
   Hidden=NO
MaxNodes=UNLIMITED MaxTime=7-00:00:00 MinNodes=0 LLN=NO
   MaxCPUsPerNode=UNLIMITED
Nodes=compute-2-[0-9]
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO
```

```
   OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=320 TotalNodes=10 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerCPU=7900 MaxMemPerNode=252800

PartitionName=compute3
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=normal
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0
   Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO
   MaxCPUsPerNode=UNLIMITED
Nodes=compute-3-0
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO
   OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=48 TotalNodes=1 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerCPU=5250 MaxMemPerNode=252000

PartitionName=compute4
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=normal
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0
   Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO
   MaxCPUsPerNode=UNLIMITED
Nodes=compute-4-[0-1]
PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO
   OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=64 TotalNodes=2 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerCPU=7900 MaxMemPerNode=252800
```

De la salida de dicho comando se puede extraer información interesante como el número de nodos de la partición (TotalNodes), los nodos que la forman (Nodes), el máximo tiempo de ejecución de un trabajo en dicha partición (MaxTime), la memoria que se asigna por defecto por core (DefMemPerCPU), etc.

La **partición por defecto** al enviar un trabajo a ejecución es **compute2** (véase el parámetro Default=YES en dicha partición). Es posible enviar un trabajo a una partición (o particiones) en concreto usando el parámetro **-p, --partition=<partition_names>**. Ejemplos de uso:

```
[root@pluton ~]# cat hostname.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
```

```
hostname
[root@pluton ~]# sbatch -t 00:10:00 -p compute0 hostname.sh
Submitted batch job 130
[root@pluton ~]# sbatch -t 00:10:00 -p compute1 hostname.sh
Submitted batch job 131
[root@pluton ~]# cat test_130.out
compute-0-0.local
[root@pluton ~]# cat test_131.out
compute-1-0.local
```

6.6 Trabajos batch paralelos

Algunos parámetros importantes para la ejecución de aplicaciones paralelas, además de las opciones -n y -c vistas anteriormente, son:

- **-N, --nodes=<minnodes[-maxnodes]>**: número mínimo y máximo (opcional) de nodos de cómputo que necesita nuestro trabajo. Si no se especifica -N, el comportamiento por defecto es asignar el número mínimo de nodos necesarios para satisfacer los requisitos solicitados mediante las opciones -n y -c.
- **--ntasks-per-node=<ntasks>**: número de tareas (procesos) que vamos a ejecutar en cada nodo de cómputo. Por defecto: --ntasks-per-node=1. Si se usa conjuntamente con -n, ésta última tiene precedencia y --ntasks-per-node se considera como el número **máximo** de tareas por nodo.

Al igual que ocurría con los parámetros -n y -c, Slurm también configura variables de entorno apropiadas para -N y --ntasks-per-node: **$SLURM_JOB_NUM_NODES** y **SLURM_NTASKS_PER_NODE**, respectivamente. Adicionalmente, la variable de entorno **$SLURM_JOB_NODELIST** contiene la lista de nodos que Slurm nos ha asignado para el trabajo actual.

Se muestra a continuación ejemplos de cómo ejecutar las aplicaciones paralelas más comunes (OpenMP, MPI, etc).

6.6.1 Aplicaciones OpenMP/Threads

Para la ejecución de trabajos paralelos que usen threads, como por ejemplo aplicaciones **OpenMP**, se puede usar el siguiente script como plantilla. Un ejemplo para la ejecución de una aplicación OpenMP usando 4 threads y solicitando 4 GiB por thread, junto con el comando para encolarlo y su salida, sería:

```
[root@pluton ~]# cat openmp.sh
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -n 1 #(una tarea/proceso en total)
#SBATCH -c 4 #(4 cores por tarea/proceso)
#SBATCH --mem-per-cpu=4G #(4 GiB por core)

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./OPENMP_hello
```

```
[root@pluton ~]# sbatch -t 00:10:00 openmp.sh
Submitted batch job 133
[root@pluton ~]# cat test_133.out
Hello World from thread = 1
Hello World from thread = 2
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 3
```

Obsérvese que **OMP_NUM_THREADS** se define con el valor correspondiente al número de cores solicitados (opción -c) usando **$SLURM_CPUS_PER_TASK**.

6.6.2 Aplicaciones MPI

Para la ejecución de trabajos paralelos con MPI se puede usar el siguiente script como plantilla. Un ejemplo usando la biblioteca Open MPI para una ejecución con 8 procesos MPI en un solo nodo, junto con el comando para encolarlo y su salida, sería:

```
[root@pluton ~]# cat mpi.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -N 1 #(1 nodo)
#SBATCH -n 8 #(8 tareas/procesos en total)

module load gnu8/8.3.0
module load openmpi3/3.1.4

mpirun -np $SLURM_NTASKS ./MPI_hello
[root@pluton ~]# sbatch -t 00:10:00 mpi.sh
Submitted batch job 140
[root@pluton ~]# cat test_140.out
compute-0-0.local: Hello World from process 7 of 8
compute-0-0.local: Hello World from process 4 of 8
compute-0-0.local: Hello World from process 5 of 8
compute-0-0.local: Hello World from process 6 of 8
compute-0-0.local: Hello World from process 3 of 8
compute-0-0.local: Hello World from process 2 of 8
compute-0-0.local: Hello World from process 1 of 8
Number of processors 8
compute-0-0.local: Hello World from process 0 of 8
```

Cabe recordar que la variable **$SLURM_NTASKS** hace referencia al número total de tareas/procesos (parámetro -n) que queremos ejecutar, en este caso 8, y que se usa como parámetro del comando **mpirun**.

Dependiendo de la implementación MPI es posible usar el comando **srun** de Slurm para ejecutar la aplicación paralela en vez de **mpirun**. Por ejemplo, con MPICH/MVAPICH2 podríamos usar el siguiente script:

```
[root@pluton ~]# cat mpi.sh
```

```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -N 1 #(1 nodo)
#SBATCH -n 8 #(8 tareas/procesos en total)

module load gnu8/8.3.0
module load mvapich2/2.3.2

srun ./MPI_hello
[root@pluton ~]# sbatch -t 00:10:00 mpi.sh
Submitted batch job 142
[root@pluton ~]# cat test_142.out
Number of processors 8
compute-0-0.local: Hello World from process 1 of 8
compute-0-0.local: Hello World from process 4 of 8
compute-0-0.local: Hello World from process 6 of 8
compute-0-0.local: Hello World from process 0 of 8
compute-0-0.local: Hello World from process 7 of 8
compute-0-0.local: Hello World from process 2 of 8
compute-0-0.local: Hello World from process 3 of 8
compute-0-0.local: Hello World from process 5 of 8
```

Destacar que en este caso no es necesario especificar el número de procesos, pues srun lo infiere del entorno. De todas formas es posible usar la opción -n de srun para especificar un número diferente de procesos si fuese necesario.

Si queremos ejecutar nuestra aplicación haciendo uso de varios nodos de cómputo, lo más sencillo es especificar el número de nodos con -N y el número de procesos MPI por nodo con **--ntasks-per-node**. Un ejemplo para ejecutar 8 procesos MPI en 2 nodos a razón de 4 procesos por nodo sería:

```
[root@pluton ~]# cat mpi.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -N 2 #(2 nodos)
#SBATCH --ntasks-per-node 4 #(4 tareas/procesos por nodo)
#SBATCH -n 8 #(8 tareas/procesos en total)

module load gnu8/8.3.0
module load openmpi3/3.1.4

mpirun -np $SLURM_NTASKS ./MPI_hello
[root@pluton ~]# sbatch -t 00:10:00 mpi.sh
Submitted batch job 143
[root@pluton ~]# cat test_143.out
Number of processors 8
compute-0-0.local: Hello World from process 0 of 8
compute-0-0.local: Hello World from process 1 of 8
```

```
compute-0-0.local: Hello World from process 3 of 8
compute-0-0.local: Hello World from process 2 of 8
compute-0-1.local: Hello World from process 4 of 8
compute-0-1.local: Hello World from process 5 of 8
compute-0-1.local: Hello World from process 6 of 8
compute-0-1.local: Hello World from process 7 of 8
```

En el ejemplo anterior sería posible omitir el parámetro -n 8 ya que Slurm lo puede calcular en base a los otros dos parámetros (-N 2 y --ntasks-per-node 4).

6.6.3 Aplicaciones híbridas MPI+OpenMP

Los trabajos paralelos híbridos utilizando la combinación **MPI+OpenMP** (o threads, en general) se pueden ejecutar usando el siguiente script como plantilla. Un ejemplo con Open MPI ejecutando 2 procesos MPI en un nodo con 4 threads por cada proceso sería:

```
[root@pluton ~]# cat hybrid.sh
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -N 1 #(1 nodo)
#SBATCH -n 2 #(2 tareas/procesos en total)
#SBATCH -c 4 #(4 cores por tarea/proceso)

module load gnu8/8.3.0
module load openmpi3/3.1.4

mpirun -np $SLURM_NTASKS -x OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK ./
   MPI_OPENMP_hello
[root@pluton ~]# sbatch -t 00:10:00 hybrid.sh
Submitted batch job 150
[root@pluton ~]# cat test_150.out
Hello from thread 1/4 from process 0/2 on compute-0-0.local
Hello from thread 2/4 from process 0/2 on compute-0-0.local
Hello from thread 1/4 from process 1/2 on compute-0-0.local
Hello from thread 0/4 from process 1/2 on compute-0-0.local
Hello from thread 3/4 from process 0/2 on compute-0-0.local
Hello from thread 0/4 from process 0/2 on compute-0-0.local
Hello from thread 2/4 from process 1/2 on compute-0-0.local
Hello from thread 3/4 from process 1/2 on compute-0-0.local
```

Obsérvese la necesidad de que la variable **OMP_NUM_THREADS** esté definida en el entorno de todos los procesos MPI que se ejecuten, y la forma de especificarlo es dependiente de cada implementación MPI. En este ejemplo, se está utilizando Open MPI que dispone del parámetro -x para exportar una variable a los procesos.

Una opción totalmente equivalente a la anterior sería especificar el número de procesos a ejecutar en cada nodo de cómputo con el parámetro --ntasks-per-node en vez de especificar el número de nodos con -N. El script equivalente quedaría:

```
[root@pluton ~]# cat hybrid.sh
#!/bin/bash
```

```
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -n 2 #(2 tareas/procesos en total)
#SBATCH --ntasks-per-node 2 #(2 tareas/procesos por nodo)
#SBATCH -c 4 #(4 cores por tarea/proceso)

module load gnu8/8.3.0
module load openmpi3/3.1.4

mpirun -np $SLURM_NTASKS -x OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK ./
   MPI_OPENMP_hello
```

Se muestra a continuación otro ejemplo ejecutando 2 procesos MPI por nodo y 2 threads por proceso utilizando 2 nodos de cómputo (por tanto, se ejecutan 4 procesos MPI en total):

```
[root@pluton ~]# cat hybrid.sh
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -N 2 #(2 nodos)
#SBATCH --ntasks-per-node 2 #(2 tareas/procesos por nodo)
#SBATCH -c 2 #(2 cores por tarea/proceso)

module load gnu8/8.3.0
module load openmpi3/3.1.4

mpirun -np $SLURM_NTASKS -x OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK ./
   MPI_OPENMP_hello
[root@pluton ~]# sbatch -t 00:10:00 hybrid.sh
Submitted batch job 151
[root@pluton ~]# cat test_151.out
Hello from thread 1/2 from process 0/4 on compute-0-0.local
Hello from thread 0/2 from process 1/4 on compute-0-0.local
Hello from thread 0/2 from process 0/4 on compute-0-0.local
Hello from thread 1/2 from process 2/4 on compute-0-1.local
Hello from thread 0/2 from process 2/4 on compute-0-1.local
Hello from thread 1/2 from process 3/4 on compute-0-1.local
Hello from thread 0/2 from process 3/4 on compute-0-1.local
Hello from thread 1/2 from process 1/4 on compute-0-0.local
```

De forma equivalente podríamos haber usado -n 4 en vez de -N 2.

6.6.4 Aplicaciones híbridas MPI+CUDA/OpenCL

La aplicaciones híbridas que utilizan MPI (o similar) junto con librerías como CUDA u OpenCL (o cualquier otra que haga uso de las GPUs) pueden ser ejecutadas utilizando el siguiente script como plantilla básica. Un ejemplo de uso junto con la orden para encolarlo y la salida del trabajo sería:

```
[root@pluton ~]# cat hybrid-gpu.sh
```

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
#SBATCH -N 2 #(2 nodos)
#SBATCH --ntasks-per-node 1 #(1 tarea/proceso por nodo)
#SBATCH --gres=gpu:K20:1  #(1 GPU K20 por nodo)

module load gnu8/8.3.0
module load openmpi3/3.1.4
module load cuda/10.1.168

mpirun -np $SLURM_NTASKS ./simpleMPI

[root@pluton ~]# sbatch -t 00:10:00 hybrid-gpu.sh
Submitted batch job 152
[root@pluton ~]# cat test_152.out
Running on 2 nodes
Average of square roots is: 0.667279
PASSED
```

En el ejemplo anterior se ha ejecutado 1 proceso MPI por nodo usando 2 nodos con 1 GPU de tipo NVIDIA K20 en cada uno de ellos.

6.7 Almacenamiento temporal

Durante la ejecución de las aplicaciones es posible hacer uso del almacenamiento **persistente** que tiene disponible cualquier usuario, ya sea $HOME (con cuotas activas) como $STORE (sin cuotas). Ambas opciones están accesibles desde todos los nodos de cómputo mediante el sistema de ficheros NFS configurado para hacer uso de la red InfiniBand.

También es posible hacer uso del almacenamiento local disponible en alguno de los sistemas de ficheros de Scratch de los nodos de cómputo (ver Sistemas de Ficheros), pero deberá ser gestionado de forma manual por los usuarios.

Adicionalmente es posible disponer de rutas para almacenamiento **temporal** que son gestionadas **automáticamente por Slurm** y que pueden ser utilizadas durante la ejecución de los trabajos, ya sea almacenamiento **local** a cada nodo de cómputo y/o almacenamiento **global compartido** por todos los nodos que forman parte de un determinado trabajo.

6.7.1 Local

Cada trabajo dispone de la variable de entorno **$LOCAL_SCRATCH** configurada con una ruta local en el sistema de ficheros de Scratch de cada nodo y que puede ser utilizada para almacenamiento temporal durante la ejecución de las aplicaciones (la variable **$TMPDIR** también se encuentra definida con el mismo valor que la anterior). Cabe destacar que dicha ruta es creada de forma automática por Slurm y que todo su contenido es eliminado cuando el trabajo termina.

```
#!/bin/bash
#
#SBATCH -J test
```

```
#SBATCH -o %x_%j.out
echo LOCAL_SCRATCH=$LOCAL_SCRATCH
echo TMPDIR=$TMPDIR
echo "Hola" > $LOCAL_SCRATCH/file
ls $LOCAL_SCRATCH
cat $LOCAL_SCRATCH/file
[rober@pluton ~]$ sbatch -t 00:10:00 script.sh
Submitted batch job 7003
[rober@pluton ~]$ cat test_7003.out
LOCAL_SCRATCH=/scratch/slurm/rober/7003
TMPDIR=/scratch/slurm/rober/7003
file
Hola
```

Algunos nodos de cómputo disponen de un sistema de ficheros de Scratch adicional alojado en un disco local de tipo **SSD** (véase Sistemas de Ficheros y Nodos compute-X-Y). Para solicitar la configuración de almacenamiento temporal en dicho disco se debe hacer uso del parámetro **-C, --constraint** especificando la opción "**SSD**". Esto configura la variable de entorno **$LOCAL_SSD_SCRATCH** con una ruta en el disco local de tipo SSD (dicha variable no será definida si no se solicita con la opción -C).

```
#!/bin/bash
#
#SBATCH -J test
#SBATCH -o %x_%j.out
echo LOCAL_SCRATCH=$LOCAL_SCRATCH
echo TMPDIR=$TMPDIR
echo LOCAL_SSD_SCRATCH=$LOCAL_SSD_SCRATCH
echo "Hola" > $LOCAL_SSD_SCRATCH/file
ls $LOCAL_SSD_SCRATCH
cat $LOCAL_SSD_SCRATCH/file
[rober@pluton ~]$ sbatch -t 00:10:00 -C SSD -p compute2 script.sh
Submitted batch job 7130
[rober@pluton ~]$ cat test_7130.out
LOCAL_SCRATCH=/scratch/slurm/rober/7130
TMPDIR=/scratch/slurm/rober/7130
LOCAL_SSD_SCRATCH=/scratch/ssd/slurm/rober/7130
file
Hola
```

6.7.2 Global

También es posible desplegar un **sistema de ficheros paralelo temporal** haciendo uso de BeeGFS [8] y su modo de funcionamiento bajo demanda denominado: **BeeOND (BeeGFS On Demand)** [9]. Esta característica de BeeGFS permite hacer uso del almacenamiento local disponible en cada nodo para desplegar un sistema de ficheros paralelo "al vuelo" que estará operativo únicamente durante la ejecución de un determinado trabajo. Por tanto, su contenido es eliminado cuando el trabajo finalice al igual que ocurría con el almacenamiento temporal local ($LOCAL_SCRATCH/$LOCAL_SSD_SCRATCH).

Es responsabilidad del usuario copiar o mover los datos a los que necesite acceder a una ruta de almacenamiento persistente como $STORE antes de terminar el trabajo.

Para solicitar el despliegue del sistema de ficheros paralelo temporal se debe hacer uso del parámetro **-C, --constraint** especificando la opción "**BeeOND**". Esto configura la variable de entorno **$GLOBAL_SCRATCH** que apunta a una ruta que representa el almacenamiento global compartido entre todos los nodos que forman parte del trabajo.

Ejemplo de uso:

```
#!/bin/bash
#
#SBATCH -J test
#SBATCH -o %x_%j.out
echo GLOBAL_SCRATCH=$GLOBAL_SCRATCH
df -h | grep beegfs_ondemand
beegfs-ctl --mount=$GLOBAL_SCRATCH --listnodes --nodetype=storage
echo "Hola" > $GLOBAL_SCRATCH/file
ls $GLOBAL_SCRATCH
cat $GLOBAL_SCRATCH/file
[rober@pluton ~]$ sbatch -t 00:10:00 -C BeeOND script.sh
Submitted batch job 7004
[rober@pluton ~]$ cat test_7004.out
GLOBAL_SCRATCH=/share/beegfs/rober/7004
beegfs_ondemand    877G   397G  481G  46% /share/beegfs/rober/7004
compute-0-0.local [ID: 1]
file
Hola
```

El espacio disponible en **$GLOBAL_SCRATCH** dependerá del espacio libre que haya en el sistema de ficheros de Scratch en los nodos involucrados en el trabajo (en el ejemplo previo se hace uso de un único nodo). Se muestra a continuación un ejemplo donde se solicitan dos nodos aumentando así el espacio disponible y el rendimiento de E/S paralelo que podríamos obtener:

```
[rober@pluton ~]# sbatch -t 00:10:00 -C BeeOND -N 2 script.sh
Submitted batch job 7004
[rober@pluton ~]$ cat test_7005.out
GLOBAL_SCRATCH=/share/beegfs/rober/7005
beegfs_ondemand    1,8T   796G  958G  46% /share/beegfs/rober/7005
compute-0-8.local [ID: 1]
compute-0-9.local [ID: 2]
```

De forma similar a otros sistemas de ficheros paralelos como Lustre, BeeGFS también permite configurar el nivel de *striping* (número de nodos entre los que se distribuye un fichero) y el tamaño de cada *stripe* [10]. Por defecto, **$GLOBAL_SCRATCH** se configura con un nivel de *striping* igual al número de nodos de cómputo del trabajo y con el stripe por defecto (512 KiB). El usuario puede cambiar dichos parámetros mediante el comando **beegfs-ctl**, pero solo puede hacerlo sobre nuevos subdirectorios creados bajo la ruta **$GLOBAL_SCRATCH**. El siguiente ejemplo aumenta el tamaño del *stripe* a 1 MiB (--chunksize=1m) manteniendo el nivel de *striping* igual al número de nodos solicitados a Slurm (--numtargets=$SLURM_NNODES):

```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out
echo GLOBAL_SCRATCH=$GLOBAL_SCRATCH
beegfs-ctl --mount=$GLOBAL_SCRATCH --listnodes --nodetype=storage
beegfs-ctl --mount=$GLOBAL_SCRATCH --getentryinfo $GLOBAL_SCRATCH
mkdir $GLOBAL_SCRATCH/data
beegfs-ctl --mount=$GLOBAL_SCRATCH --setpattern --numtargets=
   $SLURM_NNODES --chunksize=1m $GLOBAL_SCRATCH/data
echo "Hola" > $GLOBAL_SCRATCH/data/file
beegfs-ctl --mount=$GLOBAL_SCRATCH --getentryinfo $GLOBAL_SCRATCH/
   data/file
[rober@pluton ~]$ sbatch -t 00:10:00 -C BeeOND -N 2 script.sh
Submitted batch job 7011
[rober@pluton ~]$ cat test_7011.out
GLOBAL_SCRATCH=/share/beegfs/rober/7011
compute-0-8.local [ID: 1]
compute-0-9.local [ID: 2]
Entry type: directory
EntryID: root
Metadata node: compute-0-8.local [ID: 1]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 512K
+ Number of storage targets: desired: 2
+ Storage Pool: 1 (Default)
New chunksize: 1048576
New number of storage targets: 2

Entry type: file
EntryID: 1-60084EC5-1
Metadata node: compute-0-8.local [ID: 1]
Stripe pattern details:
+ Type: RAID0
+ Chunksize: 1M
+ Number of storage targets: desired: 2; actual: 2
+ Storage targets:
+ 1 @ compute-0-8.local [ID: 1]
+ 2 @ compute-0-9.local [ID: 2]
```

6.8 Trabajos interactivos

Un trabajo interactivo (o sesión interactiva) consiste en obtener un terminal en uno de los nodos de cómputo para lanzar trabajos de forma directa en él, obteniendo la salida por el propio terminal. Para iniciar una sesión interactiva en Slurm se puede utilizar el comando **srun**. Por ejemplo, para obtener una sesión de 10 minutos de duración con 2 cores y 1 GiB de memoria por core, podemos ejecutar:

```
[rober@pluton ~]$ srun -t 00:10:00 -c 2 --mem-per-cpu=1024 --pty
   bash -i
[rober@compute-0-0 pluton]$ hostname
compute-0-0.local
[rober@compute-0-0 pluton]$ echo $SLURM_JOB_ID
206
[rober@compute-0-0 pluton]$ echo $SLURM_CPUS_PER_TASK
2
[rober@compute-0-0 pluton]$ exit
exit
[rober@pluton ~]$
```

Se puede observar que una vez que el gestor planifica nuestra petición, se nos asigna un nodo de cómputo donde podremos ejecutar nuestros trabajos de forma interactiva. Para salir de la sesión interactiva se debe utilizar **exit**.

Una forma alternativa de ejecutar trabajos en interactivo sería especificando directamente a srun el comando, script o binario a ejecutar tal y como se muestra a continuación:

```
[root@pluton ~]# srun -t 00:10:00 hostname
compute-0-0.local
[root@pluton ~]# srun -t 00:10:00 date
mié oct  9 09:30:40 CEST 2019
[root@pluton ~]#
```

En el ejemplo anterior se observa que no estamos obteniendo un terminal, si no que el comando especificado es ejecutado de forma remota en uno de los nodos de cómputo obteniendo la salida por el propio terminal, y se nos devuelve el control cuando termine. Por tanto, srun utilizado de este modo es muy similar a la utilidad rsh pero bajo el control del gestor de trabajos, que será el encargado de seleccionar el nodo de cómputo adecuado (obsérvese que, al contrario de rsh, no se especifica el nodo destino donde se ejecutará el comando).

Por último, cabe destacar que también es posible conectarnos por ssh a un nodo de cómputo para ejecutar trabajos en interactivo siempre y cuando tengamos un trabajo en ejecución en dicho nodo (lanzado por ejemplo mediante sbatch). Para más información sobre esta opción véase la Sección 6.9.

6.8.1 X11 forwarding

Para la ejecución de aplicaciones que requieran hacer uso de las X es necesario habilitar el X11 forwarding en la conexión ssh al nodo *frontend*, mediante las opciones **-X**/**-Y**. Tras la conexión al *frontend* la variable de entorno DISPLAY debe estar correctamente configurada. Por ejemplo:

```
[rober@oceania ~]$ ssh -X rober@pluton.dec.udc.es
[rober@pluton ~]$ echo $DISPLAY
localhost:10.0
[rober@pluton ~]$
```

Una vez que hemos habilitado X11 forwarding en la conexión ssh al nodo *frontend*, es necesario especificar el parámetro **--spankx11** a srun, tal y como se muestra en el

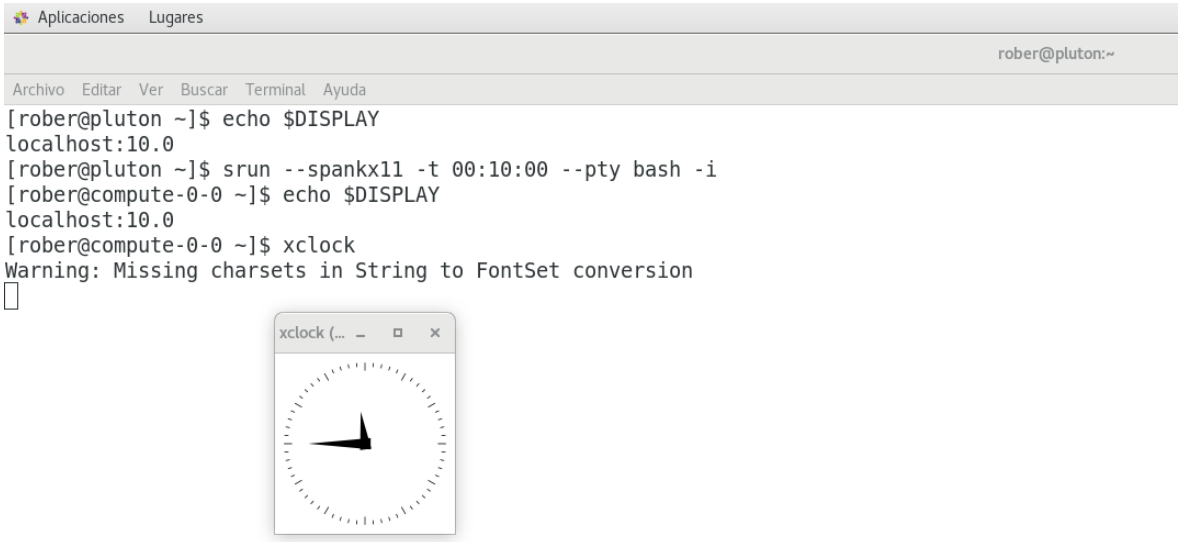

Figure 2: Trabajo interactivo con srun y X11 forwarding

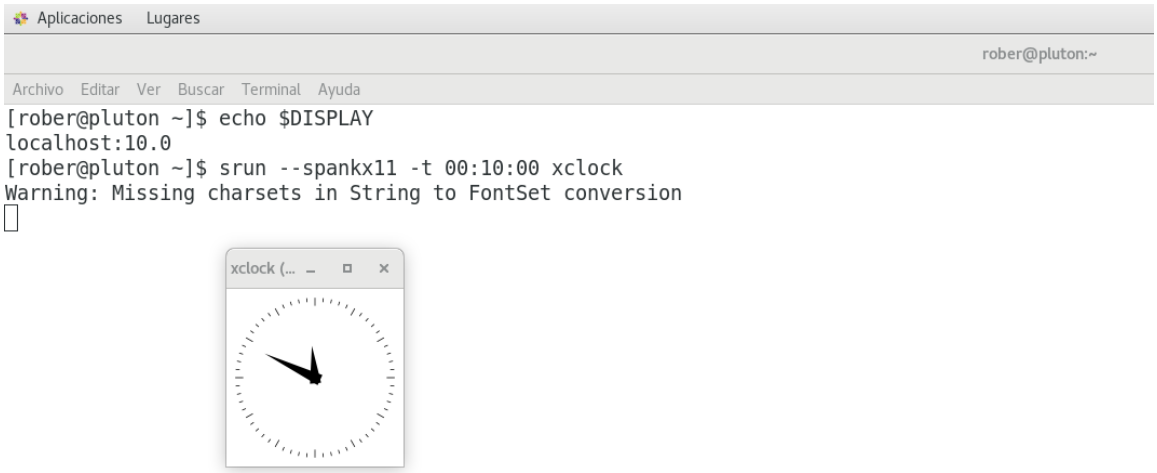

Figure 3: Trabajo interactivo con srun y X11 forwarding

siguiente ejemplo:

```
[rober@pluton ~]$ echo $DISPLAY
localhost:10.0
[rober@pluton ~]$ srun --spankx11 -t 00:10:00 --pty bash -i
[rober@compute-0-0 ~]$ echo $DISPLAY
localhost:10.0
[rober@compute-0-0 ~]$ xclock
```

En la Figura 2 se puede ver una captura de pantalla con la salida de los comandos del ejemplo anterior. La Figura 3 muestra un ejemplo de uso de srun cuando se ejecuta directamente un comando en vez de obtener un terminal.

6.9 Acceso por ssh a los nodos de cómputo

El acceso por ssh desde el nodo *frontend* a los nodos de cómputo está restringido por Slurm de forma general para impedir que un usuario pueda conectarse libremente a cualquier nodo y ejecutar aplicaciones sin el control del planificador. **Sin embargo, Slurm sí permite la conexión por ssh si un usuario tiene un trabajo en ejecución activo en el nodo al que trata de conectarse**.

En la salida de los comandos que se muestran a continuación se puede ver que no es posible inicialmente conectarse por ssh al nodo compute-0-0. Una vez que se ha enviado un trabajo *batch* y este ha entrado en ejecución, en este caso concreto en ese mismo nodo, la conexión por ssh sí es posible.

```
[rober@pluton ~]$ ssh compute-0-0
Access denied: user rober (uid=1000) has no active jobs on this
   node.
Authentication failed.
[rober@pluton ~]$ sbatch -n 2 -t 00:10:00 script.sh
Submitted batch job 250
[rober@pluton ~]$ squeue
JOBID PARTITION  NAME     USER  ST  TIME NODES  NODELIST(REASON)
250 compute0   test     rober  R  0:01     1  compute-0-0
[rober@pluton ~]$ ssh compute-0-0
Last login: Wed Oct 16 12:41:08 2019 from pluton.local
Pluton Compute Node

[rober@compute-0-0 ~]$
```

Cabe destacar que la sesión interactiva por ssh comparte los mismos recursos asignados al trabajo en ejecución en dicho nodo. Esto implica que solo es posible ejecutar tareas adicionales siempre y cuando no se superen los recursos reservados a dicho trabajo. En el ejemplo anterior se habían solicitado recursos para ejecutar dos tareas (-n 2). Dentro de la sesión ssh obtendremos un error en el momento que intentemos hacer uso de más recursos, como se ve a continuación:

```
[rober@compute-0-0 ~]$ srun -n 2 hostname
compute-0-0.local
compute-0-0.local
[rober@compute-0-0 ~]$ srun -n 4 hostname
srun: error: Unable to create step for job 250: More processors
   requested than permitted
[rober@compute-0-0 ~]$ Connection to compute-0-0 closed by remote
   host.
Connection to compute-0-0 closed.
[rober@pluton ~]$
```

También se puede observar que la sesión ssh es terminada automáticamente por Slurm cuando el trabajo *batch* que el usuario tenía en ejecución ha terminado, o bien cuando se alcance el máximo tiempo solicitado.

6.10 Ejecución de contenedores Docker

Es posible ejecutar contenedores software utilizando **udocker** [11, 12]. Esta herramienta, disponible para ser cargada como módulo (véase Lmod), permite ejecutar contenedores Docker ya que soporta su mismo formato de imagen, por lo que es posible obtener imágenes desde Docker Hub [13] y crear contenedores a partir de ellas con udocker. La principal característica de udocker es que, al contrario que Docker, permite ejecutar los contenedores software en espacio de usuario sin necesidad de privilegios de root.

Para usar udocker cada usuario del clúster dispone de su propio repositorio local y privado donde almacenar las imágenes y los contenedores. Este repositorio se crea por defecto en la ruta proporcionada por la variable de entorno **$UDOCKER_DIR**, definida apuntando a una ruta en $STORE para cada usuario ($STORE/$USER/.udocker). También es posible crear repositorios adicionales con el comando: **udocker mkrepo <dir>**.

A continuación se muestra un ejemplo de uso de udocker. El primer paso es cargar el módulo correspondiente, y la ejecución del primer comando con udocker ("udocker ps" en el ejemplo mostrado) provocará la creación del repositorio privado si no existe:

```
[rober@pluton ~]$ module load udocker/1.3.1
[rober@pluton ~]$ echo $UDOCKER_DIR
/share/store/rober/.udocker
[rober@pluton ~]$ udcoker ps
Info: creating repo: /share/store/rober/.udocker
CONTAINER ID                          P M NAMES               IMAGE
[rober@pluton ~]$ ls $UDOCKER_DIR
containers  doc  layers  repos
```

Se puede observar que el repositorio de cada usuario es un directorio oculto que reside en $STORE. Para descargar una imagen desde Docker Hub se dispone del comando **udocker pull**, aunque también es posible utilizar un Docker Registry distinto a Docker Hub usando el parámetro **--registry**. En el siguiente ejemplo se descarga una imagen de Fedora desde Docker Hub y a continuación se listan las imágenes disponibles en nuestro repositorio local mediante el comando **udocker images**:

```
[rober@pluton ~]$ udocker pull fedora
Downloading layer: sha256:
   a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
Downloading layer: sha256:
   a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
Downloading layer: sha256:9908
   e46907377e84bd6646bdb18abebeb4163b85135739e1cd60aae154d4557c
Downloading layer: sha256:
   a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
[rober@pluton ~]$ udocker images
REPOSITORY
fedora:latest
[rober@pluton ~]$ ls $UDOCKER_DIR/repos/
fedora
```

Una vez disponemos de una imagen podemos crear un contenedor a partir de ella, siendo recomendable asignarle un nombre para evitar el uso de identificadores. Se muestra

a continuación la creación de un contenedor a partir de la imagen anterior de Fedora mediante **udocker create** y se listan los contenedores disponibles mediante **udocker ps**:

```
[rober@pluton ~]$ udocker create --name=myfedora fedora
02d8e722-9eee-3a23-9d58-0578378acc8a
[rober@pluton ~]$ udocker ps
CONTAINER ID                            P M NAMES         IMAGE
02d8e722-9eee-3a23-9d58-0578378acc8a . W ['myfedora'] fedora:latest
```

Para ejecutar un contenedor se utiliza el comando **udocker run**. Por defecto este comando despliega un contenedor en el cual estaremos logueados como root, de forma que sería posible instalar software en él. Existen parámetros adicionales (**--user** y **--hostauth**) que nos permiten hacer visible nuestro usuario en el contenedor. Por otro lado, el parámetro **--volume** de **udocker run** permite montar directorios de la máquina anfitrión para que sean accesibles desde el contenedor. También podemos pasar el entorno del anfitrión al contenedor con el parámetro **--hostenv**, o montar el directorio $HOME del usuario en el contenedor con el parámetro **--bindhome**. Finalmente, para eliminar un contenedor se utiliza el comando: **udocker rm <id|name>**.

6.10.1 Ejemplos de uso

A continuación se muestra cómo crear y ejecutar un contenedor en un nodo de cómputo usando Slurm, tanto en *batch* como en interactivo. Para la ejecución en modo *batch*, un script de ejemplo podría ser el siguiente:

```
[rober@pluton ~]$ cat udocker.sh
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=%x_%j.out

module load udocker/1.3.1
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

udocker create --name=myfedora fedora
udocker run --user=rober --bindhome --hostauth --hostenv --volume=
   $STORE --volume=/share/apps --rm myfedora $HOME/OPENMP_hello
```

El comando de ejecución del contenedor especifica los parámetros necesarios para montar los sistemas de ficheros del clúster que podrían ser necesarios. De hecho, en este ejemplo el binario que ejecutará el contenedor (OPENMP_hello) reside en el directorio $HOME del usuario, con lo que debe estar accesible desde el contenedor. También se configura el acceso a $STORE a modo de ejemplo. El binario ejecutado en este caso es una sencilla aplicación paralela OpenMP, con lo que vamos a enviar el trabajo al planificador solicitando ejecutar una única tarea con 4 cores a modo de ejemplo:

```
[rober@pluton ~]$ sbatch -n 1 -c 4 -t 00:10:00 udocker.sh
Submitted batch job 17885
[root@pluton ~]# cat test_17885.out
85c73891-de67-3883-b8ad-e8e321866ff5
```

```
************************************************************
*                                                          *
*         STARTING 85c73891-de67-3883-b8ad-e8e321866ff5    *
*                                                          *
************************************************************
executing: OPENMP_hello
Hello World from thread = 0
Hello World from thread = 2
Hello World from thread = 1
Hello World from thread = 3
Number of threads = 4
```

Destacar el parámetro **--rm** del comando **udocker run** del script anterior de forma que el contenedor será eliminado automáticamente tras la ejecución del trabajo. De no especificar dicho parámetro, también podemos eliminarlo de forma manual mediante el comando **udocker rm** especificando su nombre o su identificador. Ejemplo:

```
[rober@pluton ~]$ udocker ps
CONTAINER ID                          P M NAMES         IMAGE
85c73891-de67-3883-b8ad-e8e321866ff5 . W ['myfedora'] fedora:latest
[rober@pluton ~]$ udocker rm myfedora
Info: deleting container: 85c73891-de67-3883-b8ad-e8e321866ff5
[rober@pluton ~]$ udocker ps
CONTAINER ID                          P M NAMES         IMAGE
```

Cabe destacar que también es posible crear y ejecutar un contenedor mediante **udocker run** sin necesidad de crearlo previamente con **udocker create**. Simplemente se le debe pasar como parámetro a **udocker run** el nombre de la imagen (fedora en los ejemplos previos) en vez del nombre del contenedor.

Por otro lado, si queremos ejecutar un contenedor de forma interactiva mediante srun podemos usar un comando como el del siguiente ejemplo:

```
[rober@pluton ~]$ srun -t 00:10:00 -n 1 -c 4 --pty udocker run --
   user=rober --bindhome --hostauth --hostenv --volume=$STORE --
   volume=/share/apps --rm fedora
************************************************************
*                                                          *
*         STARTING 50a5a949-e97a-390d-afb9-192d9da73aea    *
*                                                          *
************************************************************
executing: bash
50a5a949$ whoami
rober
50a5a949$ echo $HOME
/home/rober
50a5a949$ echo $STORE
/share/store/rober
50a5a949$ cat /etc/redhat-release
Fedora release 36 (Thirty Six)
50a5a949$ ls /share/apps/mpi/
mvapich2  openmpi
```

```
50a5a949$ OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK ./OPENMP_hello
Hello World from thread = 0
Number of threads = 4
Hello World from thread = 2
Hello World from thread = 3
Hello World from thread = 1
50a5a949$ exit
exit
[rober@pluton ~]$
```

Por último, otro comando importante es **udocker setup**, el cual nos permite seleccionar el modo de ejecución del contenedor mediante el parámetro **--execmode** (P1 por defecto). Para conocer todos los modos de ejecución soportados actualmente por udocker y sus diferencias se recomienda consultar su manual de usuario disponible en [14], donde además se detallan todos los comandos y parámetros de la herramienta.

Veamos un ejemplo:

```
[rober@pluton ~]$ udocker create --name=myfedora fedora
612fadc7-2ca3-3168-be18-147a4cec5e15
[rober@pluton ~]$ udocker ps -m
CONTAINER ID                          P M MOD NAMES           IMAGE
612fadc7-2ca3-3168-be18-147a4cec5e15  . W P1 ['myfedora']     fedora
[rober@pluton ~]$ udocker setup myfedora
execmode: P1
nvidiamode: False
[rober@pluton ~]$ udocker setup --execmode=P2 myfedora
[rober@pluton ~]$ udocker setup myfedora
execmode: P2
nvidiamode: False
[rober@pluton ~]$ udocker ps -m
CONTAINER ID                          P M MOD NAMES           IMAGE
612fadc7-2ca3-3168-be18-147a4cec5e15  . W P2 ['myfedora']     fedora
```

Se puede ver como después de crear un contenedor es posible obtener su modo de ejecución actual usando el parámetro **-m** del comando **udocker ps** (ver columna MOD). El comando **udocker setup** <**id**|**name**> también nos permite obtener el modo de ejecución, e incluso modificarlo mediante **--execmode**.

6.10.2 GPUs NVIDIA en contenedores

Otro parámetro interesante de **udocker setup** es **--nvidia**, el cuál habilita el acceso a GPUs NVIDIA desde el contenedor. **Para ello es necesario que la imagen Docker tenga instalada la misma versión del driver NVIDIA que el host anfitrión donde se ejecuta el contenedor**[1]. Para saber la versión que está instalada en los nodos de cómputo que disponen de GPUs NVIDIA podemos usar el comando **nvidia-smi** ejecutándolo en un trabajo interactivo solicitando una GPU:

```
[rober@pluton ~]$ srun -t 00:10:00 --gres=gpu:K20:1 nvidia-smi
```

[1] La instalación del driver NVIDIA en el contenedor se escapa del alcance que este manual pretende proporcionar.

```
Fri Jun 17 10:42:19 2022
+-----------------------------------------------------------------------+
| NVIDIA-SMI 460.56   Driver Version: 460.56   CUDA Version: 11.2 |
|-------------------------------+----------------------+----------+
```

El comando **udocker setup --nvidia** se encarga de añadir las librerías necesarias del host anfitrión al contenedor para habilitar el acceso a la GPU. Dicho comando deber ser ejecutado desde el nodo de cómputo que tiene la GPU. Por ejemplo:

```
[rober@pluton ~]$ srun -t 00:10:00 --gres=gpu:K20:1 --pty bash -i
[rober@compute-0-0 ~]$ udocker create --name=myfedora fedora
287009fc-f732-35d4-9b6b-a9104f000335
[rober@compute-0-0 ~]$ udocker setup myfedora
execmode: P1
nvidiamode: False
[rober@compute-0-0 ~]$ udocker setup --nvidia myfedora
Info: Cont, has files from previous nvidia install
Warn: nvidia file in config not found /etc/vulkan/icd.d/nvidia_icd.
   json
Warn: nvidia file in config not found /usr/bin/nvidia-installer
Warn: nvidia file in config not found /usr/bin/nvidia-uninstall
Info: nvidia mode set
[rober@compute-0-0 ~]$ udocker setup myfedora
execmode: P1
nvidiamode: True
[rober@compute-0-15 ~]$ udocker run --user=rober --bindhome --
   hostauth --hostenv --volume=$STORE --volume=/share/apps --rm
   myfedora
**************************************************************
*                                                            *
*         STARTING a2452a66-4c0c-3813-8288-53c10fcb0a8e      *
*                                                            *
**************************************************************
executing: bash
a2452a66$ nvidia-smi
Fri Jun 17 08:47:15 2022
+-----------------------------------------------------------------------+
| NVIDIA-SMI 460.56   Driver Version: 460.56   CUDA Version: 11.2 |
|-------------------------------+----------------------+----------+
a2452a66$ /share/apps/utils/simpleMultiGPU
Starting simpleMultiGPU
CUDA-capable device count: 1
Generating input data...

Computing with 1 GPUs...
GPU Processing time: 25.089000 (ms)

Computing with Host CPU...

Comparing GPU and Host CPU results...
GPU sum: 16777296.000000
```

```
CPU sum: 16777294.395033
Relative difference: 9.566307E-08
```

6.11 Avisos por correo electrónico

Es posible indicar a Slurm que nos envíe automáticamente un correo electrónico notificándonos, por ejemplo, que un determinado trabajo ha finalizado (o empezado) su ejecución. El parámetro **--mail-type** permite indicar los eventos en los se envía un correo al usuario, siendo las opciones disponibles: NONE, BEGIN, END, FAIL, REQUEUE, STAGE_OUT, ALL (equivalente a BEGIN, END, FAIL, REQUEUE, STAGE_OUT), TIME_LIMIT, TIME_LIMIT_90 (se ha alcanzado el 90% del tiempo máximo especificado), TIME_LIMIT_80, TIME_LIMIT_50, ARRAY_TASKS. El parámetro **--mail-user** permite indicar la dirección de correo electrónico a la que se enviará el correspondiente email de notificación.

Veamos un ejemplo de uso con el comando sbatch para que Slurm nos notifique que un determinado trabajo ha finalizado:

```
[rober@pluton ~]$ srun -t 00:10:00 --mail-type=END --mail-user=
   rreye@udc.es simple.sh
Submitted batch job 258
```

Cuando el trabajo finalice recibiremos un correo electrónico de notificación desde la dirección **gac.pluton.cluster@gmx.es**. El asunto del correo incluye información acerca del trabajo como su identificador, tiempo de ejecución o código de retorno. Ejemplo de asunto recibido al correo:

```
Slurm Job_id=258 Name=test Ended, Run time 00:02:01, COMPLETED,
   ExitCode 0
```

6.12 Otros comandos útiles

Otros comandos útiles a destacar de Slurm son:

- **scancel**: permite cancelar un trabajo que se encuentra en ejecución o en espera de ser planificado.
- **sacct**: permite consultar el histórico de trabajos (opción -u) y la información de un trabajo en particular (opción -j JOB_ID), entre otras opciones.
- **scontrol show job JOB_ID**: muestra información detallada sobre el estado de un trabajo en ejecución o en cola.
- **scontrol show node**: muestra información sobre el estado de los nodos de cómputo.
- **sinfo**: muestra información de nodos y particiones de forma más resumida.

7 Entorno de Software

En Pluton está instalada la suite **Rocks** [3], una distribución GNU/Linux específica para su uso en clusters HPC. La versión instalada actualmente en Pluton es **Rocks 7.0** (aka Manzanita), la cual está basada en el sistema operativo **CentOS 7.x** [4], una distribución GNU/Linux compatible a nivel binario con la distribución Red Hat Enterprise Linux (RHEL), compilada a partir del código fuente liberado por Red Hat.

Al conectarse al clúster se muestra información detallada de la versión del SO y del kernel actualmente disponibles en todos los nodos del clúster. Además, se muestra otra información interesante para el usuario como por ejemplo el número de nodos de cómputo y cores disponibles, la localización de sus directorios $HOME y $STORE y su cuota de disco, información sobre el estado de sus trabajos, etc.

```
*****************************************************
Welcome to Pluton cluster: http://pluton.dec.udc.es
******************** System Info ********************
OS version    : CentOS Linux release 7.9.2009 (Core)
Kernel        : 5.4.233-1.el7.x86_64
Uptime        : 3 days, 02h46m42s
Memory usage  : 3189/128621 MB
Load Avg.     : 4.67, 4.52, 4.41
Users logged  : 2
Avail. nodes  : 30/30 (jul 31, 2023 at 11:00:03)
Down nodes    : None
CPU cores     : 720 (jul 31, 2023 at 11:00:12)
********************* User Info *********************
$HOME  (26%) : /home/rober (quota: 4940M/40000M)
$STORE (94%) : /share/store/rober (no quota limit)
Job status    : 0/0/0 (total/pending/running)
*****************************************************
Do NOT run any computational job on the front-end
*****************************************************
```

7.1 Lmod

Pluton usa la herramienta **Lmod** [15], una versión mejorada de **Environment Modules** [16], para administrar casi todo el software con el objetivo de poder soportar diferentes versiones de un mismo paquete/biblioteca/aplicación. De esta forma, los usuarios puede cambiar de versión sin tener que especificar de forma explícita *paths* diferentes, además de que las variables de entorno son gestionadas automáticamente.

Lmod proporciona el comando **module**, cuyas opciones de uso se pueden obtener ejecutando **module help**, como se muestra a continuación:

```
[root@pluton ~]# module help
Usage: module [options] sub-command [args ...]

Options:
-h -? -H --help                    This help message
```

```
-s availStyle --style=availStyle  Site controlled avail style:
   system (default: system)
--regression_testing              Lmod regression testing
-D                                Program tracing written to stderr
--debug=dbglvl                    Program tracing written to stderr
--pin_versions=pinVersions        When doing a restore use
   specified version, do not follow defaults
-d --default                      List default modules only when
   used with avail
-q --quiet                        Do not print out warnings
--expert                          Expert mode
-t --terse                        Write out in machine readable
   format for commands: list, avail, spider, savelist
--initial_load                    loading Lmod for first time in a
   user shell
--latest                          Load latest (ignore default)
--ignore_cache                    Treat the cache file(s) as out-of
   -date
--novice                          Turn off expert and quiet flag
--raw                             Print modulefile in raw output
   when used with show
-w twidth --width=twidth          Use this as max term width
-v --version                      Print version info and quit
-r --regexp                       use regular expression match
--gitversion                      Dump git version in a machine
   readable way and quit
--dumpversion                     Dump version in a machine
   readable way and quit
--check_syntax --checkSyntax      Checking module command syntax:
   do not load
--config                          Report Lmod Configuration
--config_json                     Report Lmod Configuration in json
    format
--mt                              Report Module Table State
--timer                           report run times
--force                           force removal of a sticky module
   or save an empty collection
--redirect                        Send the output of list, avail,
   spider to stdout (not stderr)
--no_redirect                     Force output of list, avail and
   spider to stderr
--show_hidden                     Avail and spider will report
   hidden modules
--spider_timeout=timeout          a timeout for spider
-T --trace

module [options] sub-command [args ...]
```

Los subcomandos más importantes de **module** son: **list, avail, spider, load/add, unload/del, swap** y **purge**, que se detallan a continuación:

- **module list**: muestra los módulos cargados actualmente.

```
[root@pluton ~]# module list
Currently Loaded Modules:
  1) jdk/8u222  2) hwloc/2.0.3  3) git/2.22.0  4) gnu8/8.3.0
```

- **module avail**: muestra los módulos disponibles para ser cargados.

```
[root@pluton ~]# module avail
----------------- /opt/ohpc/pub/modulefiles -----------------
EasyBuild/3.9.2  cmake/3.14.3   git/2.22.0  gnu8/8.3.0
   intel_parallel_studio_xe/2019_up1  jdk/8u222  maven/3.6.1
      valgrind/3.15.0   autotools   cuda/10.1.168   gnu7/7.3.0
       hwloc/2.0.3  jdk/7u231  jdk/11.0.4    papi/5.7.0

Where:
D:  Default Module
L:  Module is loaded
```

- **module spider**: muestra **todos** los módulos disponibles en el sistema.

```
[root@pluton ~]# module spider
----------------------------------------------------------------
The following is a list of the modules currently available:
----------------------------------------------------------------
  EasyBuild: EasyBuild/3.9.2
Build and installation framework

  R: R/3.5.0, R/3.5.3
R is a language and environment for statistical computing and
   graphics (S-Plus like).

  advisor: advisor/2019_up1
A SIMD vectorization optimization and shared memory threading
   assistance tool for C, C++, C# and Fortran software
   developers

  autotools: autotools
Developer utilities

  boost: boost/1.67.0, boost/1.70.0
Boost free peer-reviewed portable C++ source libraries

  cmake: cmake/3.14.3
CMake is an open-source, cross-platform family of tools
   designed to build, test and package software.
...
```

- **module load modulefile**: carga las variables de entorno necesarias (PATH, MANPATH, LD_LIBRARY_PATH...) para el módulo seleccionado "modulefile".

```
[root@pluton ~]# module load intel_parallel_studio_xe/2019_u1
[root@pluton ~]# icc -v
icc version 19.0.1.144 (gcc version 4.8.5 compatibility)
```

- **module unload modulefile**: elimina los cambios en las variables de entorno realizadas por la carga del módulo seleccionado "modulefile".

```
[root@pluton ~]# module unload intel_parallel_studio_xe/2019
   _u1
[root@pluton ~]# icc -v
-bash: icc: command not found
```

- **module swap modulefile1 modulefile2**: descarga el módulo "modulefile1" y a continuación carga el módulo "modulefile2". Por tanto, equivale a ejecutar: **module unload modulefile1 && module load modulefile2**.

```
[rober@pluton ~]$ module list
 Currently Loaded Modules:
 1) jdk/8u222   2) hwloc/2.0.3   3) maven/3.6.1   4) git
    /2.22.0   5) gnu8/8.3.0   6) gnuplot/5.2.7
[root@pluton ~]# javac -version
javac 1.8.0_222
[root@pluton ~]# module swap jdk/8u222 jdk/11.0.4
The following have been reloaded with a version change:
1) jdk/8u222 => jdk/11.0.4
[root@pluton ~]# module list
 Currently Loaded Modules:
 1) hwloc/2.0.3   2) maven/3.6.1   3) git/2.22.0   4) gnu8
    /8.3.0   5) gnuplot/5.2.7   6) jdk/11.0.4
[root@pluton ~]# javac -version
javac 11.0.4
```

- **module purge**: elimina todos los cambios hechos por comandos previos.

```
[rober@pluton ~]$ module list
Currently Loaded Modules:
1) jdk/8u222   2) hwloc/2.0.3   3) maven/3.6.1   4) git/2.22.0
      5) gnu8/8.3.0   6) gnuplot/5.2.7
[root@pluton ~]# javac -version
javac 1.8.0_222
[root@pluton ~]# module purge
[root@pluton ~]# module list
No modules loaded
```

Por último, el usuario puede cargar módulos en su entorno por defecto simplemente añadiendo los comandos "module load modulefile" correspondientes en los ficheros $HOME/.bashrc o similares (dependiendo del shell utilizado, que por defecto es *bash*).

7.1.1 Lmod vs Enviroment Modules

La principal ventaja de **Lmod** frente al tradicional **Enviroment Modules** es su gestión **jerárquica** de los módulos, de forma que nos permite "ver" y cargar aquellos módulos que dependen de una version concreta de un compilador o biblioteca.

Cuando se carga un módulo correspondiente a un compilador (eg, gnu7/7.3.0), **module avail** nos muestra **únicamente** aquellos módulos que dependen de esa versión del compilador. De forma similar, cuando se carga una biblioteca MPI se nos mostrarán aquellos módulos que dependen de esa versión concreta de MPI. Por eso Lmod proporciona un nuevo subcomando (**spider**) el cuál nos permite ver en cualquier momento todos los módulos disponibles en el sistema independientemente del compilador/biblioteca cargada.

Esta gestión jerárquica de los módulos se puede ver en el ejemplo siguiente, donde partiendo de un entorno limpio, se carga un compilador y a continuación se observa como aparecen módulos nuevos disponibles al ejecutar **module avail**. El mismo comportamiento ocurre cuando luego se carga un módulo correspondiente a una biblioteca MPI.

```
[root@pluton ~]# module list
No modules loaded
[root@pluton ~]# module avail
------------------ /opt/ohpc/pub/modulefiles -----------------
EasyBuild/3.9.2  cmake/3.14.3   git/2.22.0  gnu8/8.3.0
   intel_parallel_studio_xe/2019_up1  jdk/8u222  maven/3.6.1
   valgrind/3.15.0   autotools   cuda/10.1.168   gnu7/7.3.0
   hwloc/2.0.3  jdk/7u231  jdk/11.0.4    papi/5.7.0

Where:
D:  Default Module
L:  Module is loaded
[root@pluton ~]# module load gnu7/7.3.0
[root@pluton ~]# module avail
------------------ /opt/ohpc/pub/moduledeps/gnu7 ------------------
R/3.5.0   gnuplot/5.2.7    gsl/2.4    hdf5/1.10.2   mpich/3.2.1
   mvapich2/2.2   openblas/0.2.20  openmpi3/3.1.0   python/3.7.4
------------------ /opt/ohpc/pub/modulefiles -----------------
EasyBuild/3.9.2  cmake/3.14.3   git/2.22.0  gnu8/8.3.0
   intel_parallel_studio_xe/2019_up1  jdk/8u222  maven/3.6.1
   valgrind/3.15.0   autotools   cuda/10.1.168   gnu7/7.3.0
   hwloc/2.0.3  jdk/7u231  jdk/11.0.4    papi/5.7.0

Where:
D:  Default Module
L:  Module is loaded
[rober@pluton ~]$ module load openmpi3/3.1.0
[rober@pluton ~]$ module avail
------------- /opt/ohpc/pub/moduledeps/gnu7-openmpi3 -------------
boost/1.67.0    fftw/3.3.7    phdf5/1.10.2    pnetcdf/1.9.0
------------------ /opt/ohpc/pub/moduledeps/gnu7 ------------------
R/3.5.0   gnuplot/5.2.7    gsl/2.4    hdf5/1.10.2   mpich/3.2.1
   mvapich2/2.2   openblas/0.2.20  openmpi3/3.1.0   python/3.7.4
------------------ /opt/ohpc/pub/modulefiles -----------------
```

```
EasyBuild/3.9.2  cmake/3.14.3   git/2.22.0  gnu8/8.3.0
   intel_parallel_studio_xe/2019_up1  jdk/8u222  maven/3.6.1
   valgrind/3.15.0   autotools   cuda/10.1.168   gnu7/7.3.0
   hwloc/2.0.3  jdk/7u231  jdk/11.0.4    papi/5.7.0

Where:
D:  Default Module
L:  Module is loaded
```

8 Ayuda

Para cualquier duda o problema con el clúster dirigirse por correo electrónico al administrador:

- Roberto R. Expósito: roberto.rey.exposito@udc.es

References

[1] Website del clúster Pluton. http://pluton.dec.udc.es

[2] Guía de usuario. http://pluton.dec.udc.es/guide/user-guide.pdf

[3] Rocks: Open-Source Toolkit for Real and Virtual Clusters. http://www.rocksclusters.org

[4] CentOS: The Community ENTerprise Operating System. http://www.centos.org

[5] Red InfiniBand. http://en.wikipedia.org/wiki/InfiniBand

[6] Slurm Workload Manager. https://slurm.schedmd.com

[7] Slurm Documentation. https://slurm.schedmd.com/documentation.html

[8] BeeGFS parallel file system. https://www.beegfs.io

[9] BeeOND: BeeGFS On Demand. https://doc.beegfs.io/latest/advanced_topics/beeond.html

[10] BeeGFS striping. https://doc.beegfs.io/latest/advanced_topics/striping.html

[11] udocker: Execution of Docker containers in user space. https://github.com/indigo-dc/udocker

[12] Gomes, Jorge, et al. Enabling rootless Linux Containers in multi-user environments: the udocker tool. Computer Physics Communications 232 (2018): 84-97.

[13] Docker Hub: Build and Ship any Application Anywhere. https://hub.docker.com

[14] udocker: User Manual https://indigo-dc.github.io/udocker/user_manual.html

[15] Lmod: A New Environment Module System. https://lmod.readthedocs.io/en/latest

[16] Enviroment Modules. http://modules.sourceforge.net