Guía de utilización de Finis Terrae II

Versiones 1

Descripción sistema 3

Portal de Usuarios 3

Conexión al sistema 3

Transferencia de ficheros y datos 3

Almacenamiento, directorios y sistemas de ficheros 3

Utilización del sistema 4

Uso del sistema de colas 5

- Comandos básicos 5
- Particiones 7
- Límites del sistema de colas (QOS) 8
- Ejemplos de envío de un trabajo al sistema de colas 9
- Ejemplos de envíos para la ejecución usando 4 cores: 11
- Asignación de la memoria 12
- Monitorización de los trabajos 12
- Salida estándar y de errores 12
- Aviso del estado del trabajo por correo 13

Algunas opciones útiles 13

Estados de los trabajos en el sistema de colas 13

Compiladores y librerías de desarrollo disponibles 14

Uso de modules/Lmod 14

Directorios Scratch 17

Uso del FAT node 17

Uso de los nodos con GPU 18

Uso de los nodos con GPU integrados del SVG 18

Uso de los nodos con Xeon PHI 18

Información adicional y ayuda 18

Problemas y dudas frecuentes 19

Anexo I. Descripción detallada del sistema 21

- Nodos Finis Terrae II 21
- Nodos integrados del SVG 22
- Nodos Finis Terrae II versus Nodos del SVG 22

Anexo II. Red Infiniband 23

Anexo III. Job Arrays 24

Anexo IV. Dependencias **25**

Anexo V. Ejecución de múltiples programas dentro de un único trabajo **26**

Ejecución de múltiples programas dentro de un trabajo con la opción --multi-prog de srun 26

Información 26

Envío del trabajo 26

Fichero de configuración 26

Trabajos multistep y opción –r,--relative 28

Ejecución de múltiples programas dentro de un trabajo utilizando GNU Parallel y srun 28

Información 28

Anexo VI. Binding: ejecución de tareas en CPUs determinados **30**

Anexo VII. Utilización de Lustre en Finis Terrae II **34**

Anexo VIII. Drop Caches **36**

Anexo IX. Ejecución de contenedores de software **37**

Uso de uDocker en Finis Terrae II 37

Creación de un repositorio local de imágenes 38

Ejecución de contenedores 39

Pull de imágenes y creación de contenedores 39

Pull desde un registry 40

Principales operaciones del comando udocker 40

Envío a cola utilizando contenedores de uDocker 41

Envío de trabajos MPI 42

Uso de Singularity en Finis Terrae II 42

Obtener y ejecutar contenedores de Singularity 43

Envío a cola utilizando contenedores Singularity 44

Anexo X. Cron redundante **46**

Versiones

Primera versión 19-2-2016
Segunda versión 22-2-2016
Tercera versión 23-2-2016
Cuarta versión 29-2-2016, incluye nueva configuración de colas
Quinta versión 1-3-2016, incluyendo directorios scratch
Sexta versión 8-3-2016, corrigiendo uso de GPUs, tips de uso de las colas
Séptima versión 15-3-2016, preguntas y respuestas
Octava versión 18-3-2016, binding de procesos
Novena versión 23-3-2016, FAT node y Lustre file striping
Décima versión, 7-4-2016, particiones
Décimo primera versión 11-4-2016, envío de mails
Décimo segunda versión 9-9-2016, documentación taller, batchlim y ejecución de múltiples tareas en un trabajo
Décimo tercera versión 20-10-2016, nuevos límites y comando myquota.
Décimo cuarta versión 23-01-2017, salida estándar y de error.
Décimo quinta versión 12-05-2017, Nueva guía de uso, reestructurada y reducida
Décimo sexta versión 26-04-2018, Cron redundante
Décimo séptima versión 18-05-2018, Modificación del apartado “Udocker”

Descripción sistema

Finis Terrae II es un sistema de computación basado en procesadores Intel Haswell e interconectado mediante red Infiniband con un rendimiento pico de 328 TFlops y sostenido en Linpack de 213 Tflops. Más detalles sobre su configuración en el Anexo I.

Portal de Usuarios

Está disponible un portal web para usuarios del CESGA a través del siguiente enlace:
https://portalusuarios.cesga.es
En donde se encuentra la siguiente información para cada usuario:

- Listado de trabajos encolados. Desde el portal se pueden realizar acciones básicas sobre estos trabajos como cancelar o ver el estado en que se encuentran
- Listado de trabajos finalizados del usuarios en los últimos días
- Espacio ocupado por el usuario en disco
- Información sobre el estado de los sistemas de cálculo, consumo de CPU o espacio utilizado en el último año
- Información sobre cómo utilizar los sistemas: envío de trabajos y simulaciones, almacenamiento disponible, aplicaciones instaladas, cómo conectarse, etc...
- Cómo obtener soporte para los problemas más habituales o abrir un caso de soporte (ticket)

Conexión al sistema

La conexión a los nodos de login del Finis Terrae II se debe realizar mediante un cliente de ssh, utilizando la siguiente dirección:

ft2.cesga.es

Utilizar el mismo usuario y contraseña del ***svg/ft*** y resto de equipos del CESGA.
Una vez conectado al sistema, se entra en uno de los nodos de login, desde los cuales se pueden realizar compilaciones, mover ficheros y enviar trabajos al sistema de colas. Estos nodos no están destinados a la ejecución de trabajos, y cada sesión está limitada en tiempo a 8 horas y 8GB de memoria. Para necesidades mayores se deberá usar un nodo interactivo (utilizando el comando **compute,** que se comenta más adelante).

Transferencia de ficheros y datos

Para la transferencia de ficheros y datos a o desde el Finis Terrae II debe utilizarse un cliente de scp o sftp.

Almacenamiento, directorios y sistemas de ficheros

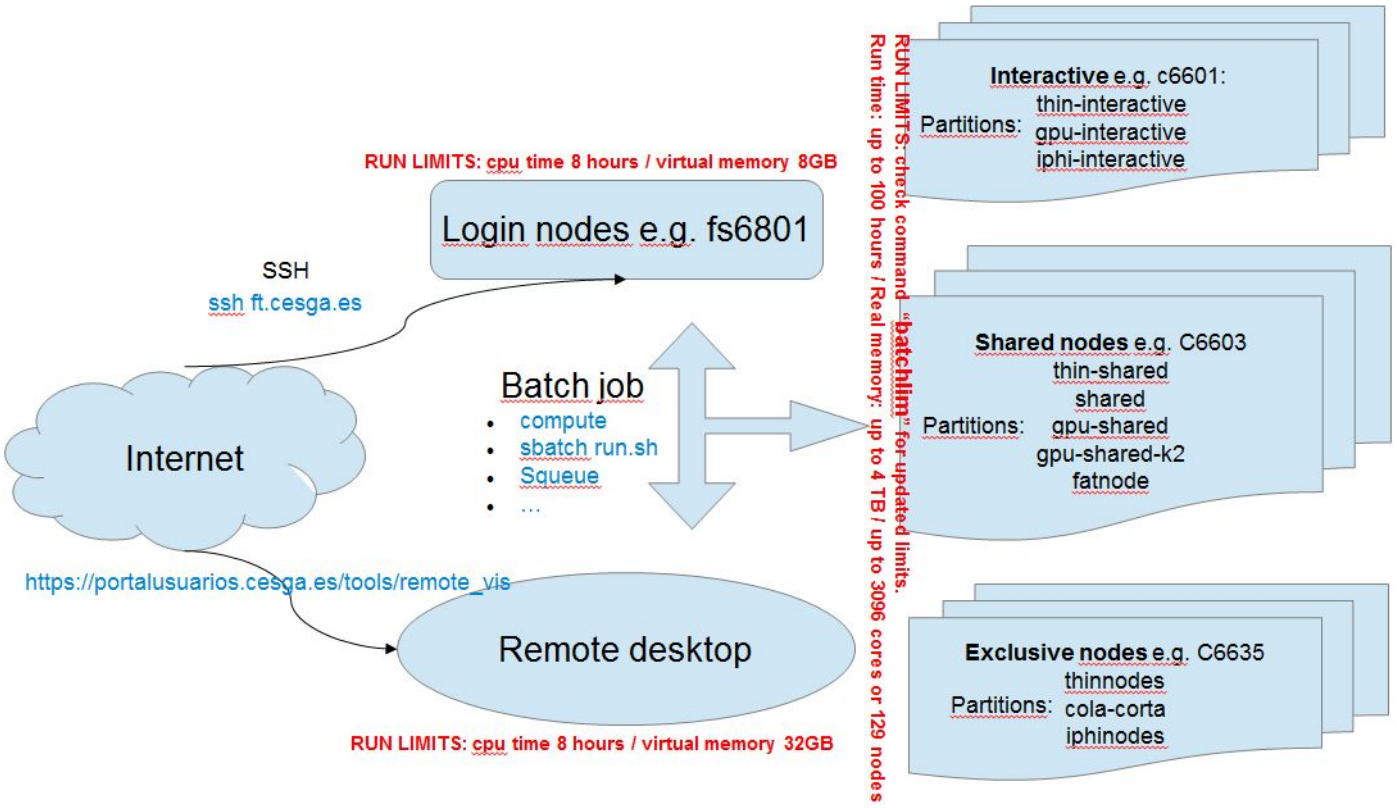
Home de FT2: $HOME, límite de 10GB, NFS, 100MB/s
Home de SVG: $HOMESVG, solo acceso de lectura, NFS, 100MB/s
Lustre: $LUSTRE, sistema de ficheros paralelo, límite de 3TB, 20GB/s
Store: $STORE, límite de 100GB, NFS, 100MB/s

Directorio / Filesystem	Propósito	Tamaño para cada usuario	Backup	Variable de entorno para acceder
Home	Almacenar ficheros de código, baja velocidad de acceso	10GB	Sí	$HOME
Store	Almacenar resultados finales de simulaciones, baja velocidad de acceso	100GB	No	$STORE
Lustre	Almacenar datos temporales de simulaciones, alta velocidad de acceso, orientado a ficheros de tamaño grande	3TB	No	$LUSTRE

Utilización del sistema

Existen cuatro formas de utilizar el equipo, dependiendo del tipo de recursos e interactividad que se necesite:

- **Uso interactivo:** al conectarse al equipo se tiene acceso a recursos limitados y compartidos, no válidos para realizar simulaciones, pero sí para hacer pequeñas compilaciones, trabajar con archivos e interactuar con el sistema. La memoria máxima disponible es de 8GB
- **Escritorio remoto:** para el acceso al escritorio remoto se puede acceder mediante un navegador a la siguiente URL: https://portalusuarios.cesga.es/tools/remote_vis a través de la cual se tiene acceso a un sistema gráfico completo X11 para realizar la visualización y acceder a aquellas aplicaciones que dispongan de un interfaz gráfico. Sin embargo tampoco será posible realizar simulaciones prolongadas ni utilizar más de 32GB de memoria.
- **Nodos interactivos dedicados:** es posible iniciar sesiones interactivas pero con recursos dedicados utilizando el comando “compute” (para más información se puede utilizar con la opción --help) desde el nodo al que se conecta el usuario. Aún así, los recursos máximos disponibles son 24 cores y 5GB de memoria por core y hasta 8 horas.
- **Utilización del sistema de colas:** para realizar simulaciones con múltiples cores y grandes cantidades de memoria y tiempo, es necesario utilizar el sistema de colas, que garantiza los recursos asignados a cada usuario. En el apartado siguiente se describe cómo se hace uso del sistema de colas.



Uso del sistema de colas

El sistema de colas está basado en SLURM. Para más información pueden consultarse las guías y tutoriales disponibles en:

Quickstart user guide: http://www.schedmd.com/slurmdocs/quickstart.html
Vídeo-tutoriales: http://www.schedmd.com/slurmdocs/tutorials.html
Taller práctico de uso del Finis Terrae II: https://aula.cesga.es/courses/COMPUTACIONCESGA/

Comandos básicos

sbatch: Envía un script a una partición de SLURM. El único parámetro obligatorio es el tiempo estimado. Así para enviar el script script.sh con una duración de 24 horas, se ejecutaría:

```
$ sbatch -t 24:00:00  script.sh
```

En caso de que el comando se ejecute con éxito, devuelve el número del trabajo (<jobid>). Ver información más detallada más adelante.

srun: Usado habitualmente para ejecutar una tarea paralela dentro de un script controlado por SLURM.

sinfo: Muestra información sobre los nodos de SLURM y las particiones. Proporciona información sobre las parciones existentes (PARTITION), si están o no disponibles (AVAIL), el tiempo máximo de cada

partición (TIMELIMIT. Si está en infinito, se regula externamente), el número de nodos que componen la partición (NODES), el estado en que se encuentra cada nodo (los más habituales son, *idle* es que está disponible, *alloc* es que está en uso, *mix* es que parte de sus CPUs están disponibles, *resv* es que se encuentra reservado para un uso específico, y *drain* es que ha sido retirado temporalmente por motivos técnicos).

Mostrar la información de una partición específica:

$ sinfo -p <partitionname>

Devolver información cada 60 segundos:

$ sinfo -i60

Mostrar las razones de por qué los nodos están: caídos, han fallado o están fallando.

$ sinfo -R

squeue: Muestra información sobre los trabajos, su estado y las colas.

Para ver el estado de mis trabajos, con <*nombre usuario*> su usuario en el sistema:

$ squeue -u <nombre usuario>

Para ver el estado de un trabajo con número <jobid>:

$ squeue -j <jobid>

Ver hora y fecha estimada de entrada en ejecución de sus trabajos pendientes:

$ squeue --start -u <nombre usuario>

Listar todos los trabajos en ejecución:

$ squeue -t RUNNING

Ver todos los trabajos en espera:

$ squeue -t PENDING

Ver todos los trabajos de un usuario en una partición:

$ squeue -p <partition name>

scancel: Permite cancelar un trabajo.

Cancelar un trabajo:

$ scancel <jobid>

Cancelar todos los trabajos pendientes :

$ scancel -t PENDING

Cancelar uno o más trabajos por su nombre:

$ scancel --name <jobname>

scontrol: Devuelve información más detallada sobre los nodos, particiones, los pasos de los trabajos y la configuración, es usado para monitorizar y modificar los trabajos en cola. Una de sus opciones más potentes es ***scontrol show jobid***, ya que es muy útil a la hora de resolver problemas puesto que nos permite ver información detallada de un trabajo.

$ scontrol show jobid -dd <jobid>

Detener un trabajo en particular (sin eliminarlo) :

$ scontrol hold <jobid>

Reanudar un trabajo detenido :

$ scontrol resume <jobid>

Volver a encolar un trabajo en particular (se cancela y se vuelve a ejecutar, equivalente a ejecutar un *scancel* y un *sbatch* seguido) :

$ scontrol requeue <jobid>

sacct: Consulta del histórico de trabajos y muestra información de consumo:

Para consultar el histórico de sus trabajos:

$ sacct -u <nombre de usuario>

Para consultar la información de un trabajo particular:

$ sacct -j <jobid>

Para mostrar todos los campos que sacct puede dar, se añade la opción "*-l*". Por ejemplo:

$ sacct -l

En caso de querer filtrar campos específicos se puede usar:

$ sacct --format=JobID,JobName,State,NTasks,NodeList,Elapsed,ReqMem,MaxVMSize,MaxVMSizeNode,MaxRSS,MaxRSSNode

Para información más detallada sobre comandos: http://slurm.schedmd.com/pdfs/summary.pdf

Particiones

Los 316 nodos de Finis Terrae II están agrupados en particiones que son conjuntos lógicos de nodos y a los que se pueden enviar los trabajos. Por defecto, todos los trabajos se envían a la partición **thin-shared**, formada por algunos nodos estándar para su uso compartido entre trabajos y pensada para trabajos que

utilizan menos cores de los disponibles en un nodo (24). Para aquellos trabajos paralelos que utilizan varios nodos, debe utilizarse la partición ***thinnodes*** formada por los nodos estándar para su utilización exclusiva por trabajos paralelos, de forma que como mucho sólo se ejecuta un trabajo en cada nodo simultáneamente. Para enviar trabajos a estos nodos, se debe utilizar la opción ***-p thinnodes*** al enviar los trabajos a cola.

Los nodos integrados del SVG están agrupados en sus propias particiones (**shared** y **gpu-shared-k2**), por lo que para usar estos nodos deberá especificar la partición deseada con la opción "***-p***" indicada anteriormente.

Límites del sistema de colas (QOS)

Los diferentes límites de recursos están asociados a QOS (Quality Of Services) diferentes. La QOS por defecto que utilizan todos los usuarios si no se especifica nada, es la QOS ***default***. Para utilizar otra QOS a la hora de enviar un trabajo, deberá utilizarse la opción ***--qos=nombre_de_la_qos***

Las QOS asociadas a los nodos del Finis Terrae II son las siguientes:

Name	Priority	MaxNodes	MaxWall	MaxJobsPU	MaxSubmitPU	MaxCPUsPU	MaxNodesPU
default	200	43	4-04:00:00	20	100		129
interactive	200		8:00:00	2	2	24	
urgent	500	4	2:00:00	5	50		
long	200	43	8-08:00:00	2	50		
extra-long	200	10	41-16:00:00	1	50		
large	200	86	4-04:00:00	2	50		
extra-large	200	256	4-04:00:00	1	50		
priority	500	43	4-04:00:00	2	50		

donde:

- **MaxNodes**: Es el número máximo de nodos que se pueden solicitar al enviar un trabajo.
- **MaxWall**: Es el tiempo máximo que se puede pedir al enviar un trabajo.
- **MaxJobsPU**: Es el número máximo de trabajos en ejecución por un usuario de forma simultánea.
- **MaxSubmitPU**: Número máximo de trabajos en cola por un usuario de forma simultánea.
- **MaxCPUsPU**: Número máximo de CPUs que puede usar un usuario de forma simultánea.
- **MaxNodesPU**: Máximo número de nodos utilizados de forma simultánea por un usuario entre todos sus trabajos en ejecución.

En el caso de los nodos integrados del SVG, las QOS existentes son las siguientes:

Name	Priority	MaxNodes	MaxWall	MaxJobsPU	MaxSubmitPU	MaxCPUsPU
shared	200	5	4-04:00:00	100	300	200
shared_long	200	1	12-12:00:00	20	300	40

shared_short	350	10	1-00:00:00	200	300	200
shared_xlong	200	1	41-16:00:00	20	300	20

En este caso, al no ser ninguna de estas QOS la asignada por defecto, deberán siempre indicar una QOS con la opción "***--qos***" al enviar sus trabajos a estos nodos.

Todos los usuarios pueden utilizar las QOS ***default***, ***interactive***, ***urgent, shared, shared_long*** y ***shared_short***. Las otras QOS deben solicitarse a través de solicitud de recursos especiales:

Estos límites pueden cambiar con el paso del tiempo, por lo que para conocer los límites existentes asociados a su cuenta puede ejecutar el comando ***batchlim***.

Ejemplos de envío de un trabajo al sistema de colas

Es obligatorio especificar el tiempo máximo de ejecución con la opción **--time=HH:MM:SS (ó de forma reducida -t HH:MM:SS)**

Utilizando srun (*no recomendable*)

```
$ srun -N2 -n2 -exclusive --time=00:00:10 hostname
```

Solicita dos servidores o nodos (N2) y envía dos tareas (n2) en modo exclusivo, (-exclusive) y ejecuta el comando "***hostname***", con un tiempo máximo de ejecución de 10 segundos. No es recomendable su uso ya que bloquea el prompt hasta la ejecución completa del trabajo.

Utilizando sbatch

Generar un script job.sh con el siguiente contenido:

```
#!/bin/sh
#SBATCH -N 2 #(solicita dos nodos)
#SBATCH -n 2 #(dos tareas en total)
#SBATCH -p thinnodes #(solicita la partición específica. Pueden solicitarse varias separadas por comas)
#SBATCH -t 00:00:30 #(30 sec ejecución)
srun hostname
```

Y enviar el trabajo con el comando:

```
$ sbatch ./job.sh
```

Las opciones de sbatch se pueden cambiar en el momento del envío. Así, para enviar el script anterior a la partición *cola-corta* sin cambiar el script, se ejecutaría:

```
$ sbatch -p cola-corta ./job.sh
```

Envío de un trabajo OpenMP

1. Compilación:

```
$ module load intel
```

```
$ ifort -qopenmp -o omphello_f omphello.f
$ icc -qopenmp -o omphello_c omphello.c
```

2. Script de ejecución en cola:

```
$ cat run.sh
#!/bin/bash
#SBATCH -n 1 #(una tarea en total)
#SBATCH -c 8 #(8 cores por tarea)
#SBATCH -t 00:10:00  #(10 min ejecución)

srun ./omphello_f
```

3. Envío a cola:

```
$ sbatch run.sh
```

Envío de un trabajo OpenMP a la partición shared

1. Script de ejecución en cola:

```
$ cat run.sh
#!/bin/bash
#SBATCH -n 1 #(una tarea en total)
#SBATCH -c 8 #(8 cores por tarea)
#SBATCH -t 00:10:00  #(10 min ejecución)
#SBATCH -p shared
#SBATCH --qos shared

srun ./omphello_f
```

2. Envío a cola:

```
$ sbatch run.sh
```

Envío de un trabajo MPI

1. Compilación del programa:

```
$ module load intel impi
$ mpiifort -o pi3 pi3.f90
```

2. Script de ejecución en cola:

```
$ cat run.sh
#!/bin/bash
#SBATCH -n 8
#SBATCH --ntasks-per-node=4
#SBATCH -c 2
#SBATCH -p thinnodes
#SBATCH -t 00:10:00
module load intel impi
srun ./pi3
```

Se solicita la ejecución en 2 nodos, usando 8 procesos (-n 8) y 4 procesos por nodo (--ntasks-per-node=4) y dos cores por proceso (-c 2, en caso de que el programa pueda usar este tipo de paralelización híbrida. Ver la siguiente sección), en total 16 cores distribuidos en 8 por nodo.. Al solicitar más de un nodo es necesario especificar en qué partición debe ejecutarse (-p thinnodes) ya que la partición por defecto admite un uso máximo de 1 nodo.
No es necesario cargar los módulos de las aplicaciones/compiladores/mpi dentro de los scripts de ejecución, basta con que estén cargados en el momento de envío a cola ya que el trabajo hereda el entorno del momento del envío. De hecho, por defecto, el directorio de trabajo será el directorio desde donde se envía el script.

3. Envío a cola:

```
$ sbatch run.sh
```

Ejemplos de envíos para la ejecución usando 4 cores:

Si se quiere reservar para una tarea **4 cores para crear 4 threads**:

```
$ sbatch -p shared --qos=shared -n 1 -c 4 script.sh
```

En caso de un ejemplo híbrido con OpenMP y MPI, en el cual se quiere reservar **2 tareas de 2 cores cada una en 2 nodos diferentes**:

```
$ sbatch -p thinnodes -n2 --task-per-node=1 -c 2 script.sh
```

Se puede observar que se está especificando que se reserven 2 nodos y 2 tareas en total (procesos MPI), que en cada nodo se ejecute una tarea y también que reserve dos cores por tarea.

En el caso de puro MPI donde cada tarea es ejecutada en un core, la reserva de **4 cores (4 tareas) en 2 nodos diferentes** sería:

```
$ sbatch -n 4 --task-per-node=2 -c 1 -p thinnodes script.sh
```

Opciones sbatch	Nº tareas (-n)	Cores por tarea (-c)	Nº nodos	Cores totales a usar	Reserva de recursos (cores reservados)
-p shared --qos=shared -n 1 --cpus-per-task=4	1	4	1	4	4 (partición compartida)
-p thinnodes -n2 --task-per-node=1 --cpus-per-task=2	2	2	2	4	48* (partición exclusiva)
-p thinnodes -n 4 --task-per-node=2 --cpus-per-task=1	4	1	2	4	48* (partición exclusiva)

* en las particiones exclusivas sólo un único trabajo puede ejecutarse en un nodo. El nº de nodos son reservados completamente independientemente de los cores que se usen

Asignación de la memoria

El slurm por defecto reserva 5GB de memoria por cada core tanto en los nodos estándar como en el fat node, pero en el caso de querer más memoria hay que especificarlo con las opciones ***--mem*** y ***--mem-per-cpu***. En el caso de un trabajo que se vaya a ejecutar en un nodo en exclusiva, por defecto se asigna toda la memoria del nodo (120GB en los nodos estándar).

A continuación se mostrarán uno cuantos ejemplos de la distribución de la memoria.

Se procede a enviar un trabajo que no necesita de paralelizar y se requiere de 6GB de memoria, se especificaría de la siguiente forma:

```
$ sbatch -n 1 --mem=6GB script.sh
```

Si lo que queremos es más de un core para una tarea, y que cada core tenga 20GB de memoria habría que hacer:

```
$ sbatch -n 1 --cpus-per-task=2 --mem-per-cpu=20GB script.sh
```

Monitorización de los trabajos

Ver trabajos en cola:

```
$ squeue
```

Cancelar un trabajo:

```
$ scancel <job_id>
```

La salida estándar del trabajo se almacena en el fichero ***slurm-<job_id>.out***

Consulta del histórico de trabajos:

```
$ sacct
```

Es posible conectarse por ssh a los nodos en los que se está ejecutando un trabajo para ver el estado del trabajo u otras tareas

Salida estándar y de errores

Slurm combina por defecto la salida estándar y de error en un único fichero llamado ***slurm-<job_number>.out*** o en ***slurm-<job_number>_<array_index>.out*** si se trata de un job array. Por defecto estos ficheros se guardan en el directorio de envío. Se puede modificar este comportamiento con las siguientes opciones del comando ***sbatch***:

--error /path/to/dir/filename
--output /path/to/dir/filename

Aviso del estado del trabajo por correo

A la hora de enviar un trabajo podemos estar interesados en saber cuando empieza, cuando termina, si ha fallado, etc, y queremos ser notificados a nuestros correos, para ello se hace con dos parámetros **--mail-type** y **--mail-user**, los cuales sirven para definir el tipo de información que queremos que nos envíe al correo y el correo al que queremos que nos llegue el aviso, el correo que nos responderá en nuestro caso será **slurm@ft2.cesga.es**

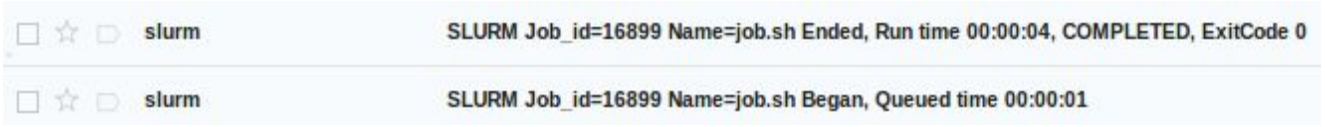
parámetros de --mail-type: BEGIN, END, FAIL, ALL, TIME_LIMIT_50, TIME_LIMIT_80, TIME_LIMIT_90, TIME_LIMIT

Ejemplo de uso en un script:

```
#!/bin/sh
#SBATCH -N 2 #(solicita dos nodos)
#SBATCH -n 2 #(dos tareas en total)
#SBATCH -t 00:00:30 #(30 sec ejecución)
#SBATCH --mail-type=begin #Envía un correo cuando el trabajo inicia
#SBATCH --mail-type=end #Envía un correo cuando el trabajo finaliza
#SBATCH --mail-user=tec_sis1@cesga.es #Dirección a la que se envía
srun hostname
```

```
# sbatch ./job.sh
Submitted batch job 16899
```

El resultado que llegará al correo tras el envío del script anterior será el mostrado aquí abajo:



Algunas opciones útiles

squeue --start: Muestra la hora estimada de comienzo de los trabajos en espera

sqstat: Muestra información detallada de las colas y la utilización global del equipo. Para utilizarlo hay que añadir las siguientes variables de entorno:

```
export STUBL_HOME=/opt/cesga/sistemas/stubl-0.0.9/
export PATH=$PATH:$STUBL_HOME/bin
```

smap: muestra un diagrama con la distribución de los trabajos en el equipo, agrupados por rack.

Estados de los trabajos en el sistema de colas

Esta es el la lista de estados en los que puede aparecer un trabajo:

- CA CANCELLED: El trabajo ha sido explícitamente cancelado por el usuario o el administrador del sistema. El trabajo puede ser que no hubiera sido iniciado.
- CD COMPLETED: El trabajo ha terminado todos los procesos en los nodos.
- CF CONFIGURING: Al trabajo se la han asignado unos recursos, pero está a la espera de ellos para poder comenzar. (e.g. booting).
- CG COMPLETING: El trabajo está en el proceso de ser completado. Algunos procesos en algunos nodos todavía están activos.
- F FAILED: Trabajo terminado con código de salida distinto de cero u otra condición de fallo.
- NF NODE_FAIL: Trabajo cancelado debido al fracaso de uno o varios nodos asignados.
- PD PENDING: El trabajo está a la espera de una asignación de recursos.
- PR PREEMPTED: Trabajo cancelado debido a un cambio en la prioridad.
- R RUNNING: El trabajo actual tiene una asignación.
- RS RESIZING: El trabajo actual está a punto de cambiar de tamaño.
- S SUSPENDED: El trabajo tiene una asignación, pero la ejecución ha sido suspendida.
- TO TIMEOUT: Trabajo cancelado por superar el tiempo límite.

Compiladores y librerías de desarrollo disponibles

La lista más actualizada de aplicaciones será la disponible mediante el comando “module spider”. Inicialmente está disponible el Intel® Parallel Studio XE 2016. El objetivo será tener disponible todas las versiones consecutivas de esta suite. Así mismo están disponibles los compiladores de GNU en diferentes versiones.
En cuanto al MPI aparte del Intel MPI estará disponible una versión actualizada de OpenMPI.

Uso de modules/Lmod

El comando module fija el entorno de variables apropiado independiente de la shell de usuario. Los comandos son idénticos al modules del svg aunque con Lmod las aplicaciones disponibles están ligadas a la carga de una combinación concreta compiladores/MPI

Módulos disponibles:

$ module avail

Inicialmente, los únicos módulos disponibles son los pertenecientes a la sección Core:

------ /opt/cesga/Lmod/lmod/lmod/modulefiles/Core -------

donde se localizan compiladores y aplicaciones/utilizadas independientes de estos.

Cargar un módulo para configurar el entorno para el uso de una aplicación:

$ module load intel/2016

```
[swtest@c6604 ~]$ module load IntelCompilers/2016
[swtest@c6604 ~]$ module av

------------------------------------------------------- /opt/cesga/Lmod/modulefiles/Compiler/IntelCompilers/2016 ----------
   bullxmpi/1.2.9.1    impi/5.1    szip/2.1

------------------------------------------------------------ /opt/cesga/Lmod/lmod/lmod/modulefiles/Core -----------------
   IntelCompilers/2016    bullxde/3.1    gcc/4.9.1    gcc/5.3.0 (D)    lmod/6.0.1    mkl/11.3    settarg/6.0.1

  Where:
   (D):  Default Module

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
```

Al cargar el módulo correspondiente a los compiladores de intel la sección de aplicaciones correspondiente a estos compiladores está disponible para la carga. Aparecen los módulos de MPI disponibles. Si se carga un módulo de MPI, por ejemplo ***bullxmpi/1.2.9.1***, aparece disponible la sección de aplicaciones disponibles para la combinación compiladores (intel) / MPI (bullxmpi).

```
[swtest@c6604 ~]$ module load bullxmpi/1.2.9.1
[swtest@c6604 ~]$ module av

---------------------------------------------- /opt/cesga/Lmod/modulefiles/MPI/IntelCompilers/2016/bullxmpi/1.2.9.1 ---------
   hdf5/1.8.16    netcdf-c/4.4.0    netcdf-fortran/4.4.3

------------------------------------------------------- /opt/cesga/Lmod/modulefiles/Compiler/IntelCompilers/2016 ----------
   bullxmpi/1.2.9.1    impi/5.1    szip/2.1

------------------------------------------------------------ /opt/cesga/Lmod/lmod/lmod/modulefiles/Core -----------------
   IntelCompilers/2016    bullxde/3.1    gcc/4.9.1    gcc/5.3.0 (D)    lmod/6.0.1    mkl/11.3    settarg/6.0.1

  Where:
   (D):  Default Module

Use "module spider" to find all possible modules.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
```

Descargar un módulo/s:

$ module unload package1 package2 ...

Módulos cargados actualmente:

$ module list

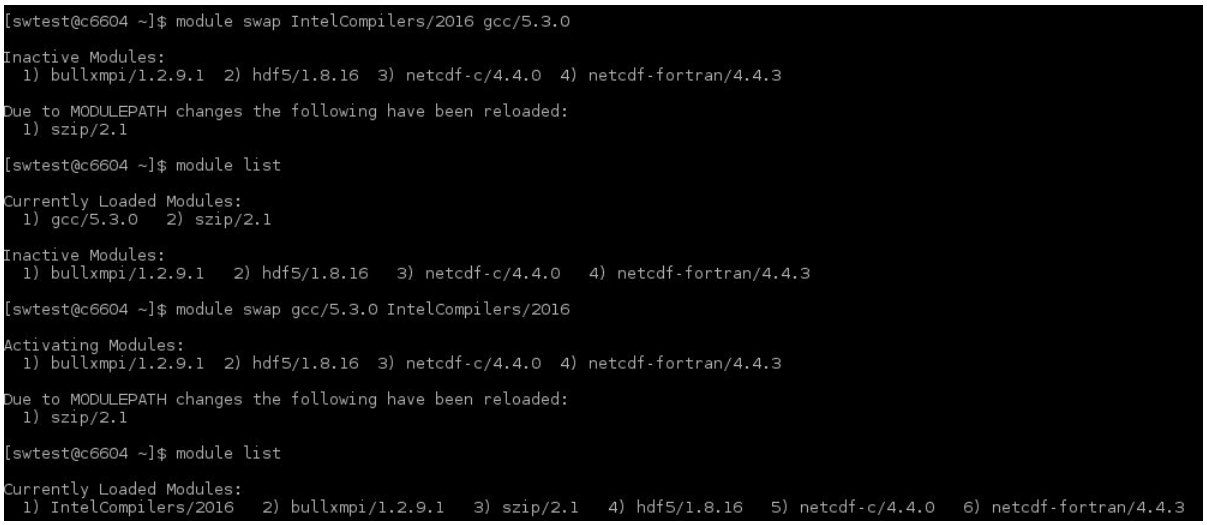
```
[swtest@c6604 ~]$ module load netcdf-fortran/4.4.3
[swtest@c6604 ~]$ module list

Currently Loaded Modules:
  1) IntelCompilers/2016   2) bullxmpi/1.2.9.1   3) szip/2.1   4) hdf5/1.8.16   5) netcdf-c/4.4.0   6) netcdf-fortran/4.4.3
```

En un momento dado puede ser interesante cambiar los compiladores en uso:

$ module swap gcc intel

Con este comando se cambiarán todas las librerías/aplicaciones dependientes del compilador. Si no existieran las aplicaciones para el compilador los módulos correspondientes se marcan con inactivos inhabilitando su uso hasta que se haga el cambio de nuevo al compilador original:



Los módulos contienen pequeñas guías de uso de las correspondiente aplicaciones/librerías. Estás se obtienen mediante el comando:

$ module help packageName

Actualmente puede haber ayudas incompletas o no disponibles.
Para obtener ayuda sobre el propio uso del comando module:

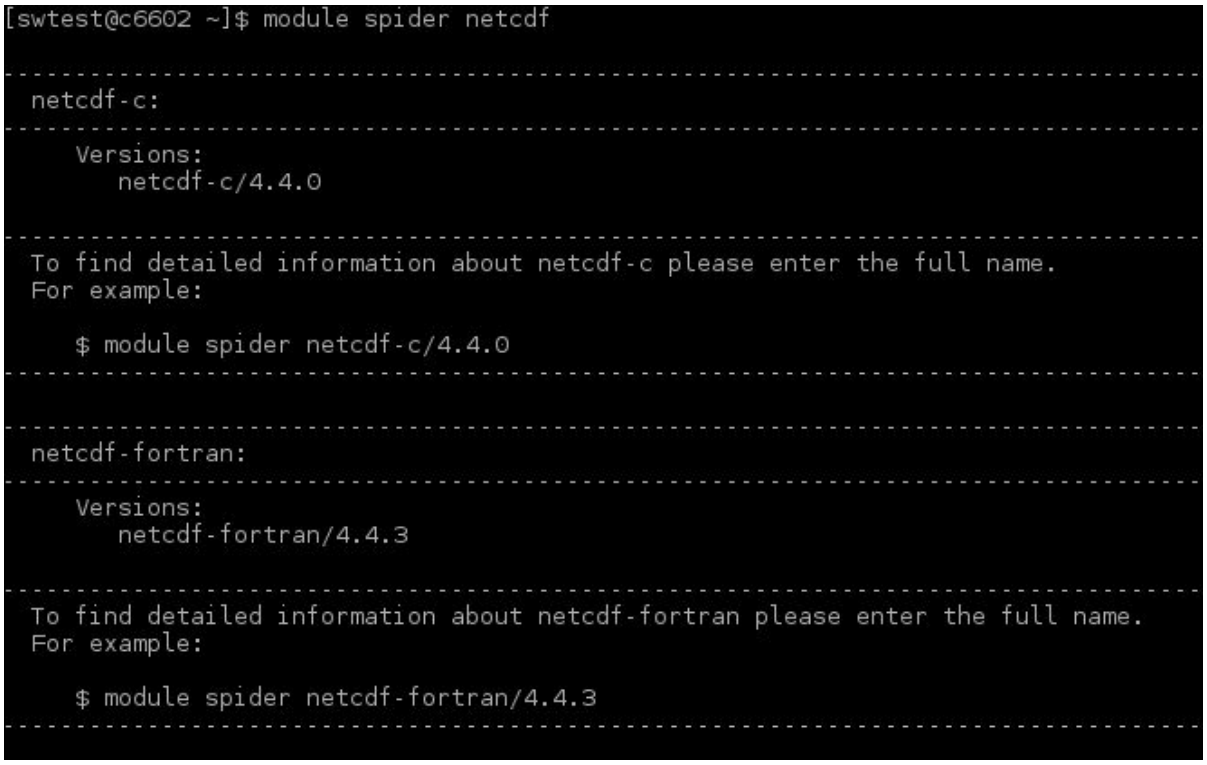
$ module help

Para ver todas las aplicaciones/librerías disponibles independientemente de los compiladores/MPI de los que dependan se usa el comando:

$ module spider

Sin ningún argumento este comando proporciona la lista completa de aplicaciones/librerías disponibles.

Es posible buscar mediante este comando si una aplicación está disponible:



Directorios Scratch

Además de utilizar directamente el directorio $LUSTRE de cada usuario, en el momento de enviar los trabajos al sistema de colas se crean dos directorios temporales que se borran automáticamente al terminar el trabajo. Estos directorios son en el caso de los nodos del FTII:

$LUSTRE_SCRATCH: directorio temporal en el almacenamiento compartido lustre.

$LOCAL_SCRATCH o **$TMPDIR**: directorio temporal en el disco local de cada servidor, no recomendada su utilización.

Nota.- Debe tener en cuenta que el espacio ocupado por los directorios de scratch del LUSTRE también se contabilizan para el cálculo de la cuota en el LUSTRE.

En el caso de los nodos integrados del SVG (partición shared), al no tener acceso directo al LUSTRE, el directorio temporal en este almacenamiento (**$LUSTRE_SCRATCH**) no está disponible por lo que solamente estará disponible el directorio temporal en el disco local (**$LOCAL_SCRATCH** o **$TMPDIR**). Por compatibilidad con los nodos del FTII, la variable **$LUSTRE_SCRATCH** se ha definido igual a **$LOCAL_SCRATCH** para estos nodos.

Uso del FAT node

El Fat node es un único nodo (con nombre interno c6714) con 128 cores y 4TB de memoria principal, así como un disco local de scratch de 20TB. Es especialmente útil en aplicaciones demandantes de grandes cantidades de memoria y que paralelizan bien utilizando modelos basados en memoria compartida (Threads, OpenMP ó Java). Para enviar trabajos a este nodo, se debe añadir la opción ***-partition fatnode*** al enviar un trabajo al sistema de colas.

Uso de los nodos con GPU

Es necesario indicar al enviar el trabajo las opciones ***--partition=gpu-shared*** y ***--gres=gpu*** (si no se indica la opción ***--gres=gpu*** no se asignará la GPU). Se puede solicitar el uso de más de una GPU con la opción ***--gres=gpu:N*** donde N es un valor entre 1 y 4. Se pueden usar hasta **6 cores por GPU**.

Ejemplo:

[carlosf@fs6801 ~]$ srun -N1 -p gpu-shared --gres=gpu -t 20 nvidia-smi topo -m

	GPU0	GPU1	GPU2	GPU3	mlx4_0	CPU Affinity
GPU0	X	PIX	SOC	SOC	SOC	0-11
GPU1	PIX	X	SOC	SOC	SOC	0-11
GPU2	SOC	SOC	X	PIX	PHB	12-23
GPU3	SOC	SOC	PIX	X	PHB	12-23
mlx4_0	SOC	SOC	PHB	PHB	X	

Leyenda:
X = Self
SOC = Path traverses a socket-level link (e.g. QPI)
PHB = Path traverses a PCIe host bridge
PXB = Path traverses multiple PCIe internal switches
PIX = Path traverses a PCIe internal switch

Uso de los nodos con GPU integrados del SVG

Es necesario indicar al enviar el trabajo las opciones ***--partition=gpu-shared-k2*** y ***--gres=gpu*** (si no se indica la opción ***--gres=gpu*** no se asignará la GPU). Se puede solicitar el uso de más de una GPU con la opción ***--gres=gpu:N*** donde N es un valor entre 1 y 2. Se pueden usar hasta **10 cores por GPU**.

Uso de los nodos con Xeon PHI

Es necesario indicar al enviar el trabajo las opciones ***--partition=iphinodes*** y ***--gres=mic***. Se puede solicitar el uso de más de un Xeon PHI con la opción ***--gres=mic:N*** donde N es un valor entre 1 y 2. Se pueden usar hasta **12 cores por Xeon PHI**.

Información adicional y ayuda

En https://aula.cesga.es/courses/COMPUTACIONCESGA/ se puede encontrar la documentación (videos y presentaciones) del ***Taller para usuarios del Finis Terrae II***.

Para cualquier problema con el acceso al sistema: *sistemas@cesga.es* o llamando al 981 56 98 10 y preguntando por el Departamento de Sistemas.

Para la parte de dudas de como compilar el programa o temas específicos de MPI: *aplicaciones@cesga.es* o llamando al 981 56 98 10 y preguntado por el Departamento de Aplicaciones.

Problemas y dudas frecuentes

P: Al conectarse por ssh al ft2 aparece el siguiente mensaje:

@@@
@   WARNING: POSSIBLE DNS SPOOFING DETECTED!   @
@@@
The RSA host key for ft2.cesga.es has changed,
and the key for the corresponding IP address 193.144.35.6
is unknown. This could either mean that
DNS SPOOFING is happening or the IP address for the host
and its host key have changed at the same time.
@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
7c:23:f3:e6:7c:f5:6d:68:58:7e:2a:41:03:69:9b:d1.
Please contact your system administrator.
Add correct host key in /xxxx/xxxxx/.ssh/known_hosts to get rid of this message.
Offending key in /exports/vm.chavez.p/.ssh/known_hosts:6
RSA host key for ft2.cesga.es has changed and you have requested strict checking.
Host key verification failed

R: Editar el fichero $HOME/.ssh/known_hosts y borrar la entrada correspondiente a ft2.cesga.es.

P: ¿Cómo puedo cambiar el número de stripes de un fichero o directorio de Lustre?

R:Por defecto, cada fichero se escribe en un único OST. Con el comando lfs es posible modificar este valor por fichero (lfs setstripe) o por directorio:

usage: setstripe [--stripe-count|-c <stripe_count>]
[--stripe-index|-i <start_ost_idx>]
[--stripe-size|-S <stripe_size>]
[--pool|-p <pool_name>]
[--block|-b] <directory|filename>

stripe_size: Number of bytes on each OST (0 filesystem default)
Can be specified with **k**, **m** or **g** (in KB, MB and GB respectively).
start_ost_idx: OST index of first stripe (-1 default).
stripe_count: Number of OSTs to stripe over (0 default, -1 all).
pool_name: Name of OST pool to use (default none).
block: Block file access during data migration.

P: ¿Es posible conectarse a los equipos en los que tengo un trabajo en ejecución a través del sistema de colas?

R: Sí, simplemente haciendo un ssh a los nodos en los que se encuentra el trabajo en ejecución. La lista de nodos en los que se está ejecutando el trabajo se puede obtener con el comando "***squeue"***

P: ¿Cómo puedo saber el espacio libre en mi directorio home?

R: Utilizando el comando ***quota*** puedes ver el espacio utilizado y el límite de utilización:

$ quota -f /mnt/EMC/Home_FT2

Disk quotas for user carlosf (uid 12345):

Filesystem	blocks	quota	limit	grace	files	quota	limit	grace
10.117.117.2:/Home_FT2								
	2309712	10240000	11264000		49572	100000	101000	

Para conocer el espacio utilizado en el LUSTRE, debe usar el comando "***lfs quota***".

Para obtener los datos de todos los directorios que tiene a su disposición puede usar el comando ***myquota***.

P: ¿Qué editores de texto están disponibles?

R:

- vim-X11 (gvim) (http://vimdoc.sourceforge.net/htmldoc/gui_x11.html): module load vim-X11
  - Pros: Muy rápido (soporta muy bien la carga de ficheros grandes de varios cientos de MB) y con toda la potencia del vi. Bien mantenido.
  - Contras: quizás complejo aunque ha mejorado mucho su uso

- nedit (https://sourceforge.net/projects/nedit) : module load nedit
  - Pros: Muy rápido (soporta muy bien la carga de ficheros grandes de varios cientos de MB). Bastante potente y personalizable.
  - Contras: Basado en librerías obsoletas (motif) y mal mantenido

- atom (https://atom.io/) : module load atom
  - Pros: Editor moderno y muy personalizable desarrollado con node.js. Bien mantenido por una comunidad grande
  - Contras: No soporta bien la carga de ficheros grandes

Anexo I. Descripción detallada del sistema

Nodos Finis Terrae II

Finis Terrae II es un sistema de computación basado en procesadores Intel Haswell e interconectado mediante red Infiniband con un rendimiento pico de 328 TFlops y sostenido en Linpack de 213 Tflops. Está compuesto por 3 tipos de nodos de computación:

306 nodos de cómputo “Thin” cada uno con:

- 2 procesadores Haswell 2680v3, 24 cores
- 128 GB memoria
- 1 disco de 1TB
- 2 conexiones 1GbE
- 1 conexión Infiniband FDR@56Gbps

6 Nodos de cómputo con Aceleradores:
4 nodos con GPUs, cada uno con:

- 2 GPUs NVIDIA Tesla K80 (2 GPUs por cada tarjeta, 4 GPUs por nodo en total)
- 2 procesadores Haswell 2680v3, 24 cores
- 128 GB memoria
- 2 discos de 1TB
- 2 conexiones 1GbE
- 1 conexión Infiniband FDR@56Gbps

(Nodos c7257-c7260)

2 nodos con Xeon Phi, cada uno con:

- 2 Intel Xeon Phi 7120P
- 2 procesadores Haswell 2680v3, 24 cores
- 128 GB memoria
- 2 discos de 1TB
- 2 conexiones 1GbE
- 1 conexión Infiniband FDR@56Gbps

(Nodos c7059 y c7060)

1 FAT node de computación con la siguiente configuración:

- 8 procesadores Intel Haswell 8867v3, 128 cores
- 4096GB memoria
- 24 discos SAS de 1,2 TB y dos discos SAS de 300 GB

(Nodo c6714)

4 nodos de login/transfer para la conexión al sistema y la transferencia de archivos, con la siguiente configuración cada uno:

- 2 procesadores Haswell 2680v3, 24 cores
- 128 GB memoria
- 2 discos de 1TB
- 2 conexiones 10Gbit Ethernet
- 1 conexión Infiniband FDR@56Gbps

Red de Interconexión de alto rendimiento Mellanox Infiniband FDR@56Gbps con topología Fat-tree

Sistema de ficheros paralelo Lustre con 768TB de capacidad (750TB netos) y 20GB/s de lectura/escritura
Consumo total de 118Kw y rendimiento pico de 328 TFlops (240 Tflops Linpack)

Area	Tipo	Modelo	Procesador	Num Nodos	Total cores	Gflops/ core	Gflops/ nodo	Tot Gflops CPU	Gflops Accel	Num Acel	Tot Gflops Accel	TOTAL Gflops
Compute	Thin	R424	E5-2680v3 12c	306	7.344	40	960	293.760				293.760
Compute	10Gb transfer IO	R423	E5-2680v3 12c	4	96	40	960	3.840				3.840
Compute	Fat	s6030	E7-8867V3 16c	4	128	40	1.280	5.120				5.120
Compute	Acel-Nvidia K80	R421	E5-2680v3 12c	4	96	40	960	3.840	1.870	8	14.960	18.800
Compute	Acel-Phi 7120P	R421	E5-2680v3 12c	2	48	40	960	1.920	1.208	4	4.832	6.752
TOTAL COMPUTO				**320**	**7.712**			**308.480**			**19.792**	**328.272**

Area	Tipo	Modelo	Procesador	Num Nodos	Total cores	Mem/nodo GBs	Tot mem/nodo GBs	Mem/GPU GBs	Tot mem GPUs	Memoria TOT GBs
Compute	Thin	R424	E5-2680v3 12c	306	7.344	128	39.168			39.168
Compute	10Gb transfer IO	R423	E5-2680v3 12c	4	96	128	512			512
Compute	Fat	s6030	E7-8867V3 16c	4	128	1.024	4.096			4.096
Compute	Acel-Nvidia K80	R421	E5-2680v3 12c	4	96	128	512	24	192	704
Compute	Acel-Phi 7120P	R421	E5-2680v3 12c	2	48	128	256	16	64	320
TOTAL COMPUTO				**320**	**7.712**		**44.544**		**256**	**44.800**

El Fat node puede ser configurado como 4 sistemas independientes o como sistema de memoria compartida con 128 cores, 4TB de memoria y con un rendimiento máximo de 1.2Tflops.

Nodos integrados del SVG

Son 54 servidores cada uno con dos procesadores Intel Haswell 2650v3 (20 cores en total), 64GB de memoria y 2 discos duros de 300GB, todos los equipos interconectados mediante dos redes Gigabit Ethernet

Además, hay otros 8 servidores cada uno con dos procesadores Intel Haswell 2650v3 (20 cores en total), 128 GB de memoria, 2 discos de 2TB y tarjeta NVIDIA GRID K2 y 4 servidores cada uno con dos procesadores Intel Haswell 2650v3, 128 GB de memoria y 2 discos de 2 TB. Los equipos están interconectados mediante 4 enlaces Gigabit Ethernet

En estos nodos, al no estar conectados mediante la red de interconexión de alto rendimiento Mellanox Infiniband, el acceso al sistema de ficheros paralelo Lustre tiene un rendimiento mucho menor. Debido a ello no existe el directorio de scratch en el Lustre.

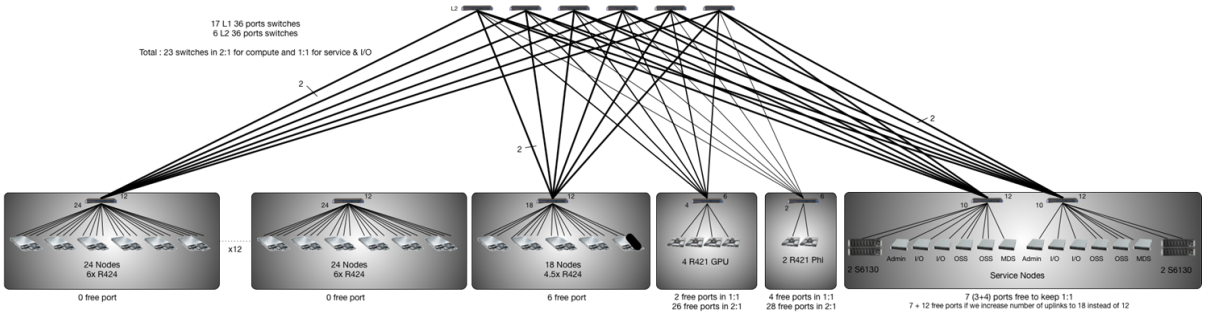
Estos nodos están accesibles a través de la partición shared del sistema de colas.

Nodos Finis Terrae II versus Nodos del SVG

Característica	Nodos FTII	Nodos SVG
Cores	24/128	20
Memoria	128G/4T	64G/128G
LUSTRE	SI. Infiniband	SI. NFS
LUSTRE_SCRATCH	SI	NO
MPI entre nodos	SI	NO

Anexo II. Red Infiniband

Todos los nodos de Finis Terrae II están conectados a una red infiniband Mellanox Infiniband FDR@56Gbps con topología Fat-tree



Anexo III. Job Arrays

Un job array en Slurm implica que el mismo script va a ser ejecutado “n” veces, la única diferencia entre cada ejecución es una simple variable de entorno, “***$SLURM_ARRAY_TASK_ID”***.
Esta opción simplifica el envío de “n” trabajos similares y le proporciona al usuario una gestión única simulando un único trabajo pero internamente slurm los trata como “n” trabajos independientes y así contribuyen en todos los límites del usuario (partición y QoS).

Opción:

-a, --array=<indexes>

Submit a job array, multiple jobs to be executed with identical parameters. The indexes specification identifies what array index values should be used. Multiple values may be specified using a comma separated list and/or a range of values with a "-" separator. For example, **"--array=0-15"** or **"--array=0,6,16-32"**. A step function can also be specified with a suffix containing a colon and number. For example, **"--array=0-15:4"** is equivalent to **"--array=0,4,8,12"**. A maximum number of simultaneously running tasks from the job array may be specified using a "%" separator. For example **"--array=0-15%4"** will limit the number of simultaneously running tasks from this job array to 4. The minimum index value is 0. the maximum value is one less than the configuration parameter MaxArraySize.

INPUT ENVIRONMENT VARIABLES

SBATCH_ARRAY_INX

OUTPUT ENVIRONMENT VARIABLES

SLURM_ARRAY_TASK_ID
SLURM_ARRAY_JOB_ID

Ejemplo: /opt/cesga/job-scripts-examples/Job_Array.sh

Anexo IV. Dependencias

A la hora de enviar los trabajos podemos indicar que el trabajo se quede en espera hasta que nosotros decidamos o bien indicarle que el trabajo depende de otros trabajos que ya estan en la cola. Las opciones a usar para estas caracteristicas son las siguientes:

- ***-H, --hold***: Indica que el trabajo será enviado en un estado retenido y deberá ser liberado por el usuario para ponerse en cola utilizando el comando "***scontrol release <job_id>***". Tambien es posible poner un trabajo en este estado una vez ya está en cola con el comando "***scontrol hold <job_id>***".

- ***-d, --dependency=<dependency list>***: Se aplaza el inicio del trabajo hasta que se cumplan todas las dependencias. <*dependency_list*> es de la forma <*type:job_id[:job_id][,type:job_id[:job_id]]*>. Varios trabajos pueden compartir la misma dependencia y estos trabajos pueden incluso pertenecer a usuarios diferente. Este valor puede cambiarse después del envío del trabajo con el comando *scontrol*.
    - ***after:job_id[:jobid...]***: El trabajo puede ejecutarse después de que los trabajos indicados hayan comenzado su ejecución.
    - ***afterany:job_id[:jobid...]***: El trabajo puede ejecutarse después de que los trabajos indicados hayan finalizado.
    - ***afternotok:job_id[:jobid...]***: El trabajo puede ejecutarse después de que los trabajos indicados hayan finalizado en error.
    - ***afterok:job_id[:jobid...]***: El trabajo puede ejecutarse después de que los trabajos indicados hayan finalizado correctamente.
    - ***singleton***: El trabajo puede ejecutarse después de que cualquier trabajo enviado previamente con el mismo nombre haya finalizado.

Anexo V. Ejecución de múltiples programas dentro de un único trabajo

Trabajos que consisten en la ejecución de varias tareas preferiblemente deberían ser combinados de alguna forma para optimizar el envío de trabajos y reducir así en número de trabajos enviados a un número más manejable.
A continuación mostramos dos formas de llevar a cabo este tipo de trabajos.

Ejecución de múltiples programas dentro de un trabajo con la opción *--multi-prog* de *srun*

Información

La opción ***--multi-prog*** del comando ***srun*** nos permite ejecutar múltiples programas dentro de único trabajo.
Se puede obtener información sobre esta opción ejecutando el comando "***man srun***" y buscando "multi-prog" en el manual mostrado.
También se puede encontrar más información en páginas como estas:

- http://www.tchpc.tcd.ie/node/167
- https://computing.llnl.gov/tutorials/linux_clusters/multi-prog.html

Ejemplo simple:
/opt/cesga/job-scripts-examples/Simple_Multiprog_Job.sh
/opt/cesga/job-scripts-examples/simple_multiprog.config

Envío del trabajo

Para enviar un trabajo con el *sbatch* necesitaremos como primer paso crear el script que vamos a utilizar. En el siguiente ejemplo vamos a solicitar la utilización de 8 cores en 1 nodo (es decir un total de 8 tareas) pero sería fácilmente adaptable al número de nodos y cores deseados utilizando las opciones necesarias del comando *sbatch*.
El script a utilizar lo llamaremos test.sh y será algo como esto:

```
#!/bin/sh
#SBATCH -n 8              # 8 cores
#SBATCH -t 10:00:00       # 10 hours
#SBATCH -p thin-shared    # partition name

srun --multi-prog test.config
```

El fichero ***test.config*** contiene los parámetros requeridos por la opción multi-prog

Fichero de configuración

El fichero de configuración contiene 3 campos, separados por espacios. Los campos son:

- El número de tarea
- El programa/script a ejecutar
- Argumentos

Parámetros disponibles:

- %t - The task number of the responsible task
- %o - The task offset (task's relative position in the task range).

Para este ejemplo utilizaremos el siguiente fichero de configuración:

```
###################################################################
# srun multiple program configuration file
#
# srun -n8 -l --multi-prog test.config
###################################################################
0 /home/cesga/prey/multiprog/script.sh %t
1 /home/cesga/prey/multiprog/script.sh %t
2 /home/cesga/prey/multiprog/script.sh %t
3 /home/cesga/prey/multiprog/script.sh %t
4 /home/cesga/prey/multiprog/script.sh %t
5 /home/cesga/prey/multiprog/script.sh %t
6 /home/cesga/prey/multiprog/script.sh %t
7 /home/cesga/prey/multiprog/script.sh %t
```

***Nota**.- Nótese que en el caso de utilizar un ejecutable/script propio como en este caso, se deberá indicar la ruta completa al fichero.*

***Nota**.- El número de tareas indicado en este fichero de configuración ha de ser exactamente el mismo que el número de tareas solicitadas mediante sbatch.*

En este ejemplo, se indica que queremos ejecutar un script (*script.sh*) que requiere el uso de un core y al cual se le pasa como argumento el número de tarea, el cual podemos utilizar de modo similar a como se hace con los job-arrays para identificar algún parámetro necesario en la ejecución.
Es posible indicar que el script utilice más de un core para su ejecución utilizando la opción "*-c NCORES*" y además es posible indicar diferentes programas/scripts dentro de este fichero de configuración en donde para cada entrada se utilice un script diferente. En la información indicada al inicio de esta sección tiene ejemplos de ello.
Para el ejemplo mostrado anteriormente, dado que se trata de ejecutar el mismo script para todas las tareas podríamos simplificarlo poniendo una única línea como la siguiente:

```
0-7 /home/cesga/prey/multiprog/script.sh %t
```

Si utilizamos por ejemplo el siguiente script:

```
#!/bin/sh

echo "TAREA=$1"
```

obtendríamos la siguiente salida en la cual se puede ver el identificador de cada tarea:

```
TAREA=7
TAREA=0
TAREA=1
TAREA=4
TAREA=5
TAREA=3
TAREA=2
TAREA=6
```

***Nota**.- El orden en el que salen reflejadas las tareas puede ser diferente entre ejecuciones del mismo trabajo.*

Trabajos multistep y opción –r,--relative

Ejecución de diferentes programas a la vez en distintos recursos dentro de un trabajo. Varios srun pueden ejecutarse al mismo tiempo dentro de un trabajo concreto siempre que no superen los recursos reservados para ese trabajo:

```
#!/bin/bash
#SBATCH -n 8                    # 8 tasks
#SBATCH --ntasks-per-node=4     # 2 Nodes
#SBATCH -t 10:00                # 10 min
#SBATCH -p thinnodes,cola-corta # partition

#STEP 0
srun -n4 sleep 60 &
#STEP 1
srun -n4 sleep 60 &
wait
```

En este caso las 4 tareas del step 0 y las 4 tareas del step 1 se ejecutarán concurrentemente indistintamente en los 2 nodos asignados al trabajo. Sin embargo con la siguiente modificación:

```
#STEP 0
srun -N1 -n4 sleep 60 &
#STEP 1
srun -N1 -r1 -n4 sleep 60 &
```

las tareas del step 0 se ejecutarán en el primer nodo y las del segundo step en el segundo nodo. Nótese la necesidad de especificar -N1 para definir el nº de nodos donde se ejecutarán las tareas del step 0 dentro de los recursos del trabajo y -N1 -r1 en el caso del step 1 para definir el nº de nodos y la posición relativa de estos nodos dentro de los recursos asignados al trabajo.

-r, --relative=<n>

Run a job step relative to node n of the current allocation. This option may be used to spread several job steps out among the nodes of the current job. If -r is used, the current job step will begin at node n of the allocated nodelist, where the first node is considered node 0. The -r option is not permitted with -w or -x option and will result in a fatal error when not running within a prior allocation (i.e. when SLURM_JOB_ID is not set). The default for n is 0. If the value of --nodes exceeds the number of nodes identified with the --relative option, a warning message will be printed and the --relative option will take precedence.

Ejemplo combinado multiprog/multistep: **/opt/cesga/job-scripts-examples/Multiprog_Job.sh**

Ejecución de múltiples programas dentro de un trabajo utilizando *GNU Parallel* y *srun*

Información

Cuando el número de tareas a realizar es muy grande y dichas tareas no duran todas tiempos similares, la opción anterior no parece la más adecuada sino que sería más conveniente la utilización conjunta de *GNU Parallel* y *srun*. Con este método de envío, se enviará un único trabajo con un número elegido de ctareas (usando la opción *--ntasks=X*) y el comando ***parallel*** se utilizará para ejecutar ese número de tareas simultáneamente hasta que todas las tareas se hayan completado. Cada vez que acabe una tarea, otra será puesta en marcha sin tener que esperar a que finalicen todas.
Puede encontrar más información en la siguiente página:
https://rcc.uchicago.edu/docs/running-jobs/srun-parallel/index.html#parallel-batch

Independientemente de la partición y recursos solicitados en el trabajo las opciones srun y GNUparallel recomendadas son:

```
MEMPERCORE=$(( $(sinfo -p $SLURM_JOB_PARTITION -N -o "%m/%c" -h) ))
SRUN="srun --exclusive -N1 -n1 -c1 --mem-per-cpu=$MEMPERCORE"

# --delay .2 prevents overloading the controlling node
# -j is the number of tasks parallel runs so we set it to $SLURM_NTASKS
# --joblog makes parallel create a log of tasks that it has already run
# --resume makes parallel use the joblog to resume from where it has left off
# the combination of --joblog and --resume allow jobs to be resubmitted if
# necessary and continue from where they left off
parallel="parallel --delay .2 -j $SLURM_NTASKS --joblog logs/runtask.log --resume"

# this runs the parallel command we want
# in this case, we are running a script named runGP.sh
# parallel uses ::: to separate options. Here {0..99} is a shell expansion
# so parallel will run the command passing the numbers 0 through 99
# via argument {1}
$parallel "$SRUN ./runGP.sh {1} > logs/parallel_{1}.log" ::: {0..99}
```

Estas opciones usan parallel para ejecutar múltiples srun simultáneos. En estas opciones se contempla la ejecución de tareas secuenciales (uso de un único core) pero es factible también la ejecución de tareas multicore ó multiproceso ajustando adecuadamente la variable de entorno SRUN:

```
SRUN="srun --exclusive -N1 -n1 -c4 --mem-per-cpu=$MEMPERCORE" (multicore con 4 cores)
SRUN="srun --exclusive -N1 -n4 -c1 --mem-per-cpu=$MEMPERCORE" (multiproceso con 4
procesos)
SRUN="srun --exclusive -N1 -n4 -c4 --mem-per-cpu=$MEMPERCORE" (híbrido con 4 procesos y 4
cores por proceso)
```

La opción --exclusive siempre es necesaria en la ejecución de distintos job steps dentro de un mismo nodo ya que asegura el uso único de los recursos por cada tarea.

Ejemplo:
/opt/cesga/job-scripts-examples/GNUParallel.sh
/opt/cesga/job-scripts-examples/runGP.sh

Anexo VI. Binding: ejecución de tareas en CPUs determinados

En las arquitecturas multinúcleo actuales para obtener un rendimiento óptimo es recomendable siempre usar opciones de "***binding***" de procesos a CPUs (cores) físicos no permitiendo al sistema la migración de procesos entre los distintos cores y asegurando la mayor proximidad entre los datos en memoria y el core que los lee/escribe. Por defecto en la configuración del FT2 cada una de las tareas solicitadas están asociadas a un cgroup (Linux Control Groups) que no permitirá que los procesos de la tarea se ejecuten fuera de los recursos físicos asociados a esa tarea.

Definiciones:
Topología del nodo por defecto (128GB RAM y 24 cores):

Cada socket dispone de 64GB RAM y 12 cores. Destacar que el disco local reside en el socket 0 y las conexiones de infiniband (lustre y comunicaciones MPI) en el socket 1.
Procesador lógico: Procesador a nivel de sistema operativo (software). Por defecto en el FT2 los procesadores lógicos corresponden a cores físicos. En algún momento puede haber nodos con el "***Hyperthreading***" activado (2 o más threads por core), en ese caso el procesador lógico corresponde a un thread de uno de los cores. Esta configuración está por defecto para los coprocesadores Xeon Phi donde hay 4 threads por core físico.

Afinidad: Asignación de un proceso a un determinado procesador lógico.

Máscara de afinidad: Máscara de bits donde los índices corresponden a procesadores lógicos. '1' implica posibilidad de ejecución de un proceso en el procesador lógico asociado a la correspondiente posición en la marca.

Con slurm tenemos las siguientes posibilidades:

1) **Afinidad a nivel de tarea (Puro MPI)**. Siempre es recomendable especificar el número de tareas por nodo "***--ntasks-per-node***" o el sistema de colas asignará un reparto en modelo bloque donde se intenta llenar el nodo antes de usar el siguiente asegurando siempre por lo menos una tarea por

nodo.
Con la opción:

--cpu_bind=[{quiet,verbose},]type

es posible especificar la afinidad de cada una de las tareas/procesos a procesador lógico. A continuación se describe las opciones de type recomendadas.

type:

rank: Afinidad automática según el rango (rank) MPI del proceso. La tarea con rango más bajo en cada nodo es asignada al procesador lógico 0.
Ejemplo:

```
srun -t 00:10:00 -N2 -n8 --ntasks-per-node=4 --cpu_bind=verbose,rank ./pi3
cpu_bind=RANK - c7202, task  4  0 [16381]: mask 0x1 set
cpu_bind=RANK - c7202, task  5  1 [16382]: mask 0x2 set
cpu_bind=RANK - c7202, task  6  2 [16383]: mask 0x4 set
cpu_bind=RANK - c7202, task  7  3 [16384]: mask 0x8 set
cpu_bind=RANK - c7201, task  1  1 [17456]: mask 0x2 set
cpu_bind=RANK - c7201, task  2  2 [17457]: mask 0x4 set
cpu_bind=RANK - c7201, task  3  3 [17458]: mask 0x8 set
cpu_bind=RANK - c7201, task  0  0 [17455]: mask 0x1 set
...
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       17455    c7201    +1
[0] MPI startup(): 1       17456    c7201    +1
[0] MPI startup(): 2       17457    c7201    +1
[0] MPI startup(): 3       17458    c7201    +1
[0] MPI startup(): 4       16381    c7202    +1
[0] MPI startup(): 5       16382    c7202    +1
[0] MPI startup(): 6       16383    c7202    +1
[0] MPI startup(): 7       16384    c7202    +1
...
```

Los 8 procesos MPI son asignados a los procesadores lógicos 0,1,2 y 3 (socket 0) de cada uno de los nodos. Esto no es una situación recomendable teniendo en cuenta que la infiniband (comunicaciones MPI) reside en el socket 1.

map_cpu:<list>: afinidad por nodo a la lista de procesadores lógicos especificados. El ejemplo anterior debería ejecutarse mejor especificando está afinidad:

```
$ srun -t 00:10:00 -N2 -n8 --ntasks-per-node=4 --cpu_bind=verbose,map_cpu:12,13,14,15 ./pi3
cpu_bind=MAP  - c7201, task  1  1 [18684]: mask 0x2000 set
cpu_bind=MAP  - c7201, task  0  0 [18683]: mask 0x1000 set
cpu_bind=MAP  - c7201, task  2  2 [18685]: mask 0x4000 set
cpu_bind=MAP  - c7201, task  3  3 [18686]: mask 0x8000 set
cpu_bind=MAP  - c7202, task  4  0 [17234]: mask 0x1000 set
cpu_bind=MAP  - c7202, task  5  1 [17235]: mask 0x2000 set
cpu_bind=MAP  - c7202, task  6  2 [17236]: mask 0x4000 set
cpu_bind=MAP  - c7202, task  7  3 [17237]: mask 0x8000 set
...
```

Nótese en el cambio de la máscara de afinidad (hexadecimal).
Se recomienda el uso de esta opción ya que es la que da un control y una versatilidad mayor.

Existen dentro de ***"srun"*** opciones extra dedicadas a la especificación de la distribución de tareas:

--distribution=arbitrary|<block|cyclic|plane=<options>[:block|cyclic|fcyclic]>

El primer argumento especifica la distribución de las tareas en los nodos y el segundo el "***binding***" de los procesos.
Es recomendable consultar la página del man de ***"srun"*** respecto a esta opción por si pudiese facilitar la ejecución de un caso concreto pero debido a la situación de los buses de infiniband en el socket 1 no se recomienda su uso por defecto ya que todas las políticas comienzan en el socket 0.

2) **Afinidad a nivel de thread (trabajos OpenMP y MPI/OpenMP).**
La afinidad de los threads derivados de procesos paralelizados en memoria compartida (OpenMP en general) se especifica a través del uso de máscaras de afinidad (nº en hexadecimal que convertido en binario la posición de los "1" habilita el uso del procesador lógico).

Como el caso de la ejecución de los trabajos MPI o serie al sistema de colas se debe solicitar el número de tareas a ejecutar así como el número de procesadores lógicos (cores en el FT2) asignados a esa tarea mediante la opción:

-c, --cpus-per-task=<ncpus>

También con la opción:
--cpu_bind=[{quiet,verbose},]type
es posible especificar la afinidad de cada una de las tareas/procesos a procesador lógico pero en este caso en type debe especificarse opciones de máscara.

type:
mask_cpu:<list>

Con esta opción se especifica la máscara de cada una de las tareas a ejecutar por nodo. Se aconseja usar una máscara en hexadecimal compuesta de 6 dígitos (cada dígito especifica la máscara de ejecución de 4 cores, 24 cores en total, 12 primeros corresponden al socket 0 y 12 segundos al socket 1) y ella especificar exclusivamente 0 o "f" habilitando o no el uso de los correspondientes 4 cores. Ejemplos de máscaras:

f00f00: Los threads se ejecutarán en los 4 primeros cores del socket 0 y en los 4 primeros del socket 1
000fff (ó fff): Los threads se ejecutarán en los 12 cores del socket 0
fff000: Los threads se ejecutarán en los 12 cores del socket 1

Toda la combinatoria de máscaras posibles para los nodos de 24 cores se puede generar con esta aplicación javascript:

https://vis.cesga.es/mask_affinity/mask_affinity.html

Ejemplos de ejecuciones con ***"srun"*** (se usa la utilidad cpuinfo de Intel mpi como programa ejemplo a ejecutar ya que da información de los procesadores lógicos asignados):

srun -t 00:10:00 -p thinnodes -n1 -c8 --cpu_bin=verbose,mask_cpu:f0000f cpuinfo -d
srun -t 00:10:00 -p thinnodes -n2 -c8 --cpu_bin=verbose,mask_cpu:f0000f,00ff00 cpuinfo -d
srun -t 00:10:00 -p thinnodes -n4 --tasks-per-node 2 -c4 --cpu_bin=verbose,mask_cpu:f0000f,00ff00 cpuinfo -d

Se recomienda exclusivamente usar las opciones de binding disponibles en slurm. No se recomienda mezclar las opciones de binding de los distintos paquetes: usar opciones de binding con slurm y a su vez usar las proporcionadas por las librerías MPI o de paralelización en memoria compartida (OpenMP).

Anexo VII. Utilización de Lustre en Finis Terrae II

Finis Terrae II utiliza Lustre para su directorio **$LUSTRE_SCRATCH**. En el caso de muchas aplicaciones, la utilización de striping de ficheros puede mejorar el rendimiento de E/S a disco. Esta técnica mejora especialmente el rendimiento de las aplicaciones que realizan E/S serie desde un único servidor o que hace E/S paralela desde múltiples servidores escribiendo en un único fichero compartido con MPI-IO, HDF5 paralelo o NetCDF paralelo.

El sistema de ficheros Lustre está formado por un conjunto de servidores de E/S (OSSs) y de discos (OSTs). Un fichero se puede repartir entre varios OSTs y entonces se denomina striping. De esta forma se mejora el rendimiento de entrada salida al acceder a varios OSTs de forma simultánea, incrementando el ancho de banda de E/S.

El striping por defecto en Finis Terrae II es de 1, lo que quiere decir que por defecto sólo se utiliza 1 OST para cada fichero, independientemente de su tamaño. El tamaño de un stripe es de 1MB.

Determinar el striping óptimo para un fichero no es fácil, ya que si se realiza sobre pocos OSTs no sacará partido del ancho de banda disponible, mientras que si se hace sobre demasiados, provocará un overhead innecesario y pérdida de rendimiento. Para más información pueden consultarse a sistemas@cesga.es. La tabla siguiente muestra una recomendación sobre cómo configurar este striping en función del tamaño del fichero:

Tamaño del fichero	E/S compartida entre todos los procesadores	Un fichero por procesador
<1GB	No hacer nada. Utilizar striping por defecto	No hacer nada. Utilizar striping por defecto
1GB-10GB	4 OSTs	No hacer nada. Utilizar striping por defecto
10GB-100GB	12 OSTs	No hacer nada. Utilizar striping por defecto
100GB-1TB	48 OSTs	Preguntar

El stripe de un fichero debe fijarse antes de empezar a escribir en él. Por ejemplo, para crear el fichero “salida” que más adelante puede crecer hasta 100GB, se utilizará el comando:

lfs setstripe --count 12 salida

Que fijaría el stripesize del fichero salida en 12 OSTs

El mismo comando se puede utilizar para generar un directorio y especificar el striping por defecto de todos los ficheros contenidos en él. Por ejemplo, el siguiente comando crea el directorio dir_salida con un striping de 12 para todos los ficheros que se generen dentro de él:

lfs setstripe --count 12 dir_salida

El striping de un fichero no se puede cambiar una vez que se ha comenzado a escribir en el fichero (a menos que se copia a un nuevo fichero vacío creado con el striping deseado)

Para conocer el stripe de un fichero se puede utilizar el comando:

lfs getstripe fichero

IMPORTANTE: se debe minimizar o evitar la utilización del comando "***ls -l"*** en los directorios de Lustre, ya que provoca una carga en el sistema. También debe evitarse la creación de directorios con muchos ficheros, ya que cada vez que se abren varios ficheros de un mismo directorio, se provoca una contención. En ese caso es preferible tener varios subdirectorios con menos ficheros. También debe evitarse almacenar los ficheros pequeños y ejecutables en Lustre, y mover esos ficheros al directorio home o store.

Anexo VIII. Drop Caches

Al final de cada trabajo de la partición thinnodes se ejecuta un drop caches que limpia la cache de los nodos para evitar caídas del rendimiento entre trabajos.

Anexo IX. Ejecución de contenedores de software

En Finis Terrae II es posible la ejecución de contenedores de software a través de las aplicaciones uDocker y Singularity. Un contenedor de software consiste en un conjunto formado por la aplicación, sus dependencias, librerías y archivos de configuración que se puede ejecutar sobre el anfitrión. A diferencia de una máquina virtual, un contenedor se ejecutará sobre la máquina anfitrión compartiendo el kernel con él. Por lo tanto su tamaño será mucho menor que el de una máquina virtual y su ejecución y despliegue será más rápido.

uDocker utiliza el formato de imágenes de Docker que se pueden obtener, por ejemplo, desde DockerHub. Esta posibilidad permite la ejecución de aplicaciones que de manera nativa no podrían ejecutarse debido a incompatibilidades o ausencia de librerías en la máquina anfitrión. Entre otras funcionalidades, uDocker permite mapear directorios de la máquina anfitrión para que sean accesibles desde el contenedor.

Singularity también es compatible con los contenedores de Docker, pero Singularity trabaja con su propio formato y, por lo general, es necesaria una conversión previa. Tanto la creación como la conversión de contenedores debe hacerse con privilegios de superusuario, por lo que NO se puede en Finis Terrae II. El proceso habitual será la creación del contenedor en la máquina del usuario y su posterior copiado a su cuenta de Finis Terrae II. Singularity también provee de una herramienta para el almacenamiento y descarga de contenedores públicos, SingularityHub.

Uso de uDocker en Finis Terrae II

uDocker se encuentra instalado a disposición de todos los usuarios y se puede cargar como un módulo,

```
$ module load udocker/1.1.1
```

También existe disponible la versión 1.0.0. Una vez cargado, podemos consultar la ayuda general de la aplicación con el comando **help**. Los comandos son similares a los de Docker

```
$ udocker help
```

Por defecto, cuando se carga el módulo, se apunta al repositorio de imágenes gestionado por el CESGA. Este repositorio está en una ruta del sistema que no tiene permisos de escritura para los usuarios regulares, así que las imágenes que se encuentran en él sólo se pueden utilizar, pero no modificar. Más adelante veremos cómo crear un repositorio de imágenes en alguno de nuestros sistemas de archivos, lo cual nos permitirá descargar imágenes, almacenarlas y crear contenedores de uso personal.

Para listar las imágenes disponibles en el repositorio del CESGA a partir de las cuales podríamos crear un contenedor:

```
$ udocker images
```

Por ejemplo, en este momento están disponibles las siguientes:

```
REPOSITORY
feelpp/feelpp-toolboxes:latest                    .
openporousmedia/opmreleases:latest                .
```

```
quay.io/fenicsproject/stable:1.6.0          .
tensorflow/tensorflow:latest-gpu            .
tensorflow/tensorflow:latest                .
victorsndvg/salome-v7.8.0:nvidia            .
```

Para consultar los contenedores disponibles en el repositorio del CESGA, usamos el comando **ps**

```
$ udocker ps
```

El cual nos devolverá una lista con los contenedores que podemos ejecutar por defecto,

```
CONTAINER ID                 P M NAMES          IMAGE
39d36fd9-8175-39d5-a620-d23614f8bb06 . R ['tensorflow-0121-gpu'] tensorflow/tensorflow:latest-gpu
4a5d9cd9-8682-35db-8459-903a49510d39 . R ['opm-2017_04']        openporousmedia/opmreleases:latest
4fa3d9f1-6198-30e4-8681-cbc00b904d25 . R ['tf2']            tensorflow/tensorflow:0.12.0-rc0
54253cfe-5bce-3b0a-8e87-ac31db761a84 . R ['tf-gpu']         tensorflow/tensorflow:latest-gpu
5be91b67-15eb-3962-9193-47cfdfdc6e4f . R ['salome-7_8_0']   victorsndvg/salome-v7.8.0:nvidia
71a214cd-3de9-348f-a660-bfbe12c2a943 . R ['tf2-gpu']        tensorflow/tensorflow:latest-gpu
73c09df7-2c4e-3395-a8f3-09a9a755c9f6 . R ['fenics-1_6_0']   quay.io/fenicsproject/stable:1.6.0
b697b467-22dd-3afb-a8d4-2ccbc36207a6 . R ['feelpp-toolboxes'] feelpp/feelpp-toolboxes:latest
ce6ea3b4-7a3e-3b49-a6db-cb55f64575d0 . R ['tf']             tensorflow/tensorflow:latest
```

Creación de un repositorio local de imágenes

Como mencionamos anteriormente, cuando se carga el módulo udocker, por defecto apunta a un repositorio de imágenes y contenedores gestionado por el CESGA. Dado que existe la posibilidad de importar imágenes del docker-hub y crear contenedores a partir de ellas, se hace necesario crear un repositorio en alguno de nuestros sistemas de archivos, bien sea el $HOME, $STORE, $LUSTRE, o alguno otro que tengamos disponible. Esto es debido a que el repositorio del CESGA no tiene permisos de escritura para usuarios regulares y por ello no permitirá que escribamos en él. La ruta del repositorio del CESGA se puede consultar haciendo un echo de la variable de entorno UDOCKER_DIR

```
$ echo $UDOCKER_DIR
/opt/cesga/udocker/localrepo
```

Para crear un repositorio local se utiliza el comando **mkrepo**, por ejemplo, si queremos situarlo en nuestro directorio $HOME en la carpeta “repo_local” haríamos lo siguiente,

```
$ udocker mkrepo $HOME/repo_local
```

Sin embargo, **puede ser interesante utilizar $STORE como lugar donde colocar nuestro repositorio privado**, debido a que cada contenedor genera una gran cantidad de ficheros y puede llegar a llenar la cuota del $HOME con rapidez.

Una vez creado el repositorio, debemos apuntar a él, ya que por defecto udocker apunta al del CESGA,

```
$ export UDOCKER_DIR=$HOME/repo_local
```

O en el caso de utilizar el filesystem $STORE,

```
$ export UDOCKER_DIR=$STORE/repo_local
```

Es importante resaltar que **no podemos apuntar simultáneamente** a dos repositorios, debemos seleccionar uno de ellos únicamente.

Ejecución de contenedores

La ejecución de un contenedor por defecto se hace con el comando **run**

```
$ udocker run [options] <container-id-or-name>
```

Por ejemplo, si quisiéramos ejecutar el contenedor cuyo nombre es **'tf2-gpu'** podríamos hacerlo de dos formas distintas, a través de su "ID" o a través de su nombre,

```
$ udocker run 71a214cd-3de9-348f-a660-bfbe12c2a943
$ udocker run tf2-gpu
```

La ejecución de los anteriores comandos provocará que se despliegue el contenedor en el cual estaremos logueados como usuario root, esto es útil para realizar instalaciones dentro de él.

El siguiente ejemplo permitiría ejecutar una shell de bash en el contenedor tf2 como usuario cesga_user y montar los tres principales sistemas de archivos de Finis Terrae II ($HOME, $LUSTRE Y $STORE),

```
$ udocker run --user=<usuario> --volume=$HOME --volume=$LUSTRE --volume=$STORE /
--workdir=$HOME tf2 /bin/bash
```

Sin embargo, ya que en numerosas ocasiones es útil mapear los diferentes filesystems a los que un usuario puede acceder, así como a directorios donde las aplicaciones pueden buscar librerías en el anfitrión, el equipo técnico del CESGA ha automatizado el proceso de ejecución con el mapeo de directorios y filesystems a través de un script que es ejecutable una vez cargado el módulo udocker, este script se denomina **run_udocker.sh**. La ejecución de un contenedor a través de este script se haría de la siguiente forma, por ejemplo para desplegar un contenedor y obtener una terminal de bash:

```
$ run_udocker.sh tf2-gpu /bin/bash
```

Para cada comando, podemos consultar la ayuda específica acompañando con la opción **--help**, por ejemplo, para el comando "run",

```
$ udocker run --help
run: execute a container
run [options] <container-id-or-name>
[...]
```

Pull de imágenes y creación de contenedores

Una vez creado el repositorio local de imágenes, es posible escribir en él y almacenar allí las imágenes que podemos obtener del Docker-hub. Por ejemplo, podemos buscar en imágenes de fedora,

```
$ udocker search fedora
$ udocker pull fedora
```

Una vez hecho esto podemos crear contenedores a partir de esta imagen utilizando el comando **create**, para el cual es recomendable utilizar la opción **--name** para asociarle un nombre identificativo,

```
$ udocker create --name=primer_contenedor fedora
```

Pull desde un registry

Es posible utilizar un registry distinto al DockerHub, por ejemplo, para hacer un pull de un RH7 desde el de Red Hat:

```
$ udocker pull --registry=https://registry.access.redhat.com rhel7
```

A partir de este podríamos crear nuestro contenedor de Red Hat 7

```
$ udocker create --name=rh7 rhel7
$ udocker run rh7
```

Principales operaciones del comando **udocker**

A continuación mostramos una lista de los principales comandos de udocker

Syntax:

udocker <command> [command_options] <command_args>

Donde <command> puede ser:

- **search**
  Busca en el Docker Hub imágenes de contenedores
- **pull**
  Hace un pull de una imagen desde el repositorio de docker, que por defecto es el dockerhub. Las siguientes opciones están disponibles
  - **--index=url** Especifica un index diferente a index.docker.io
  - **--registry=url** Especifica un registro distinto a resgistry-1.docker.io
  - **--httpproxy=proxy**
- **create**
  Crea un contenedor desde una imagen del repositorio local. Se recomiendata utilizar la opción **--name** para asignarle un nombre fácil de recordar.
- **run**
  Ejecuta un contenedor. En el siguiente apartado veremos un **punto importante sobre los diferentes modos de ejecución.** Ya que esto afectará al rendimiento de posibles aplicaciones.
- **images**
  Lista las imágenes disponibles en el repositorio local.
- **ps**
  Lista los contenedores creados.
- **rm**
  Elimina un contenedor previamente creado a partir de una imagen.
- **rmi**
  Elimina una imagen de un contenedor previamente descargada o importada. Con la opción **-f** se fuerza la eliminación aunque ocurran errores durante el borrado
- **inspect**
  Muestra la información sobre los metadatos de un contenedor, acepta también como input el ID de la imagen de la que proviene.
- **import**

Importa un tarball desde un archivo. Puede ser utilizado para importar un contenedor que ha sido exportado utilizando **docker export** creando una nueva imagen en el repositorio local.

  - Con la opción **--tocontainer** se crea el contenedor sin crear una imagen intermedia
  - Con la opción **--clone** se importa un contenedor creado con udocker (**udocker export --clone**) sin crear una imagen intermedia.

- **load**

Carga en el repositorio local un tarball que contiene una imagen de Docker, es equivalente a hacer un pull desde el Docker Hub, pero en este caso se carga la imagen desde un archivo. Es la opción que utilizaríamos para cargar una imagen guardada con **docker save**.

- **protect/unprotect**

Protege una imagen o un contenedor de borrados accidentales

- **mkrepo**

Crea un repositorio local en el directorio especificado DIRECTORY, por defecto, cuando se carga el módulo utiliza una ruta protegida contra escritura que debe ser modificada para crear un repositorio privado de imágenes y contenedores.

- **login**

Para hacer login en un Docker registry. Solo soporta autentificación con username y password.

- **setup**

Esta opción es de **suma importancia** para ejecución de aplicaciones multihilo. Debido a limitaciones en la funcionalidades de la versión del sistema operativo disponible actualmente en el Finis Terrae II, determinadas aplicaciones que utilizan varios hilos de computación pueden ver su rendimiento afectado, en cuyo caso ha de modificarse el **modo de ejecución** del contenedor mediante la opción **setup**. La forma de modificación es la siguiente,

$ udocker setup --execmode=<modo> CONTAINER-ID | CONTAINER-NAME

Donde el modo recomendado para la ejecución de aplicaciones multihilo en caso de rendimiento degradado sería **F2**, a continuación ponemos una lista de modos recomendados,

Mode	Engine	Description	Changes container
P1	PRoot	accelerated mode using seccomp	No
P2	PRoot	seccomp accelerated mode disabled	No
F1	Fakechroot	exec with direct loader invocation	symbolic links
F2	Fakechroot	F1 plus modified loader	F1 + ld.so

- **clone**

Esta opción duplica un contenedor creando una réplica completamente exacta, pero recibiendo un CONTAINER-ID diferente. Es recomendable renombrarla utilizando la opción **--name=NAME**

Y como mencionamos anteriormente, cada comando tiene su ayuda específica añadiendo la opción "--help".

Envío a cola utilizando contenedores de uDocker

En esta sección veremos cómo realizar una ejecución mediante el uso de contenedores utilizando el sistema de colas para enviar el contenedor a un nodo de cálculo. Utilizando el siguiente script de ejemplo podemos hacernos una idea de cómo sería el script que arranca el contenedor cuando el trabajo entra en ejecución.

En el siguiente ejemplo vemos un trabajo que solicita un nodo y 24 tareas, un tiempo de ejecución de 5 minutos y ser enviado a la partición thinnodes, en el que desplegará el contenedor “rh7” y ejecutará el comando “hostname”

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 24 #(24 tareas en total)
#SBATCH -t 00:05:00 #( 5 min ejecucion )
#SBATCH -p thinnodes

module load udocker/1.1.1
# Si queremos usar nuestro repo local apuntaremos a la dirección correcta
export UDOCKER_DIR=$STORE/repo_local
/opt/cesga/udocker/1.1.1/udocker run  --hostenv --hostauth --user=<usuario> -v /mnt -v /tmp -v=$HOME
--workdir=$HOME rh7 hostname
```

Si tenemos la necesidad de ejecutar una serie de comandos lo más adecuado es crear un script con ellos en un directorio que montemos cuando se ejecute el contenedor y ejecutarlo de la siguiente manera:

```
/opt/cesga/udocker/1.1.1/udocker run  --hostenv --hostauth --user=<usuario> -v /mnt -v /tmp -v=$HOME
--workdir=$HOME rh7 /bin/bash comandos.sh
```

Donde **comandos.sh** será un script con aquello que queremos ejecutar a través del contenedor, por ejemplo,

```
# !/bin/bash
echo $PWD >> /home/cesga/$USER/prueba_$USER
ls -l >> /home/cesga/$USER/prueba_$USER
cat /etc/os-release >> /home/cesga/$USER/prueba_$USER
hostname >> /home/cesga/$USER/prueba_$USER
```

Y simplemente enviaríamos el script que llama a uDocker con sbatch.

Envío de trabajos MPI

Es posible realizar ejecuciones con Open MPI y udocker utilizando Slurm en Finis Terrae II. Cada uno de los procesos MPI será un contenedor. Este mecanismo es ejecutado por el mpiexec del anfitrión y estos contenedores serán capaces de comunicarse a través de la interfaz Infiniband del anfitrión.

Los requisitos para poder realizar estas ejecuciones son; el código en el contenedor ha de ser compilado con la misma versión del MPI disponible en el anfitrión. Esto es un requisito indispensable para que la ejecución funcione, por lo tanto, ha de descargarse dentro del contenedor la versión de Open MPI que se vaya a utilizar en Finis Terrae II y ser compilada dentro.

A mayores, los paquetes **openib** y **libibvers** tienen que ser instaladas para poder compilar Open MPI sobre infiniband. Una descripción completa de este procedimiento puede ser consultado en el **apartado 4 “Running MPI Jobs”** de la documentación oficial.

Uso de Singularity en Finis Terrae II

Singularity está accesible para los usuarios desde el sistema de módulos del Finis Terrae II:

```
$ module load singularity/2.4.2
```

Para comenzar a trabajar con Singularity, el primer paso es consultar la ayuda que proporciona la aplicación:

```
$ singularity --help
```

También puedes encontrar la respuestas a muchas preguntas frecuentes en la sección FAQ de su página oficial. Además, encontrarás información más detallada en sección Guía de usuarios de esa página.

A diferencia de Docker o uDocker, la aplicación Singularity no gestiona un repositorio de imágenes. Cada imagen es un único fichero que se almacena en el sistema de ficheros del anfitrión. Las imágenes de Singularity, al contener sistemas operativos completos, pueden llegar a ser grandes, por lo que recomendamos almacenarlas en **$STORE**, en donde tendrás una cuota de espacio mayor que en **$HOME**.

Obtener y ejecutar contenedores de Singularity

Para obtener imágenes desde el registro oficial de Singularity se utiliza el comando "**pull**". La referencia a las imágenes sigue la siguiente sintaxis "**URI://Collection/Imagen:tag**", en el que "**shub://**" es la URI que referencia al registro oficial de Singularity. Además mediante la opción "**--name**" se puede especificar la ruta y el nombre de la archivo destino en el sistema de ficheros local. Singularity se basa en los principios de portabilidad y reproducibilidad, por lo que, por defecto, las imágenes de Singularity son inmutables, archivos que contienen un sistema de ficheros SquashFS comprimido y de solo lectura.

```
$ singularity pull --name openmpi-ring-1.10.simg shub://MSO4SC/Singularity:ring_1.10.7
```

Para solicitar información o inspeccionar una imagen de singularity se puede utilizar los comando "**help**", "**inspect**" o "**apps**". El comando "**help**" muestra un mensaje de ayuda sobre el contenedor, mientras que "**inspect**" y "**apps**" te ayudan a descubrir las características internas y las aplicaciones instaladas en el mismo.

```
$ singularity help openmpi-ring-1.10.simg
$ singularity inspect openmpi-ring-1.10.simg
$ singularity apps openmpi-ring-1.10.simg
```

Para ejecutar aplicaciones dentro de un contenedor Singularity se utilizan los comandos "**run**", "**exec**" o "**test**". Tanto "**run**" como "**test**" ejecutan aplicaciones predefinidas en el momento de creación de la imagen. El comando "**exec**" permite ejecutar un comando personalizado dentro del contenedor. Las variables de entorno del anfitrión están por defecto accesibles desde el contenedor.

```
$ singularity run openmpi-ring-1.10.simg
$ singularity test openmpi-ring-1.10.simg
$ singularity exec openmpi-ring-1.10.simg /usr/bin/ring
```

Al ejecutar comandos desde un contenedor es importante discernir qué pasa dentro o fuera del contenedor. Fíjate que a la hora de escribir datos utilizas directorios del anfitrión (no del contenedor) en los que tienes permisos. Por defecto, desde un contenedor Singularity tienes acceso a los directorios **$HOME**, **/tmp** y **/var/tmp** del anfitrión. Para tener acceso a todos tus directorios en el Finis Terrae II, lo más sencillo

es indicarle al contenedor que utilice **/mnt** del anfitrión mediante la opción "**-B**", así tendrás acceso a los volúmenes a los que accedes normalmente desde tu cuenta de usuario del Finis Terrae II (**$LUSTRE**, **$STORE**, etc.). La opción "**-B**" permite montar cualquier cualquier directorio del anfitrión en el contenedor, pero ten en cuenta que **la ruta de destino debe existir dentro del contenedor**.

```
$ singularity exec -B /mnt openmpi-ring-1.10.simg /usr/bin/ring
$ singularity exec -B /mnt openmpi-ring-1.10.simg ls $STORE
```

Además, puedes utilizar el comando "**shell**" de Singularity para interaccionar con el contenedor de forma interactiva. Cuando ejecutas este comando te devuelve un nuevo prompt y cualquier comando que escribas se ejecutará dentro del contenedor. Por defecto utiliza una shell Bourne (**sh** o **/bin/sh**). Para salir del contenedor y volver al nodo de login simplemente escribe "**exit**". Al ser interactivo, el comando "**shell**" no está indicado para ser utilizado en trabajos enviados a cola.

```
$ singularity shell -B /mnt openmpi-ring-1.10.simg
```

Envío a cola utilizando contenedores Singularity

Para enviar trabajos a cola utilizando Singularity debes recordar la carga del módulo correspondiente. El envío de trabajos secuenciales o con hilos no tienen ninguna particularidad. Puedes crear un script SBATCH como habitualmente para describir la ejecución de tus trabajos con Singularity. Puedes ver un ejemplo a continuación:

```
#!/bin/bash
#SBATCH -p thin-shared
#SBATCH -n 1
#SBATCH -N 1
#SBATCH -t 00:01:00
module load singularity/2.4.2
singularity exec openmpi-ring-1.10.simg ring
```

También se pueden ejecutar aplicaciones MPI con contenedores Singularity. Existe una restricción que impone que la librería de MPI dentro del contenedor debe coincidir con la del anfitrión (en fabricante y versión) para asegurar el funcionamiento. En el caso de OpenMPI puedes consultar la versión del contenedor con un comando como el siguiente:

```
$ singularity exec ring_ompi_1.10.simg ompi_info | grep "Open MPI"
```

Una vez te asegures de que coincide con alguna versión del anfitrión y de cargarla mediante el sistema de módulos, podrás enviar tus trabajos MPI como habitualmente. En el caso de trabajos que reserven una gran cantidad de nodos puede ser interesante almacenar tu imagen en **$LUSTRE** para obtener mejores ratios de transferencia. Ten en cuenta que OpenMPI necesita tener acceso al directorio **/scratch** para gestionar algunos archivos temporales y que el contenedor puede no contener ese directorio. En ese caso debemos indicar otro, como por ejemplo **/tmp** que se utiliza en el siguiente ejemplo:

```
#!/bin/bash
#SBATCH -p thinnodes
#SBATCH -n 48
```

```
#SBATCH -N 2
#SBATCH -t 00:01:00
module load gcc/5.3.0 openmpi/1.10.2 singularity/2.4.2
mpirun -np 2 -mca orte_tmpdir_base /tmp singularity exec ring_ompi_1.10.simg ring
```

Para ejecutar tus aplicaciones GPU dentro de un contenedor, se puede utilizar la opción "**--nv**" a los comandos de ejecución de Singularity. Esta es una opción experimental que permite utilizar los drivers de NVIDIA dentro del contenedor. Para que esto funcione correctamente, el software del contenedor debe haber debe haber sido compilada con soporte GPU de NVIDIA y **debe contener el entorno de NVIDIA** ("**nvidia-smi**"). A continuación se muestra un script de ejemplo.

```
#!/bin/bash
#SBATCH -p gpu-shared-k2
#SBATCH --gres=gpu
#SBATCH -n 1
#SBATCH -N 1
#SBATCH -t 00:01:00
singularity/2.4.2
singularity exec --nv IMAGEN COMANDO
```

Anexo X. Cron redundante

Cuando te conectas al Finis Terrae II, accedes a uno de los 4 nodos de login disponibles. Normalmente, cada vez que te conectes, accederás al mismo nodo de login pero esto puede cambiar, por ejemplo, si accedes desde sitios diferentes (desde tu centro y desde casa), o si el nodo al que solías acceder no está disponible en ese momento.

Supongamos que necesitas configurar una tarea en el cron para que se ejecute cada cierto tiempo. En este caso, tendrías que editar tu cron con el comando "***crontab -e***" y adecuarlo a tus necesidades. Una vez hecho esto, tu tarea se ejecutará en el momento indicado en el nodo de login en el cual lo hayas configurado.

Nota.- El acceso al cron está restringido por lo que aquellos usuarios que quieran utilizarlo deberán solicitarlo enviando un mail a sistemas@cesga.es.

Si la siguiente vez que te conectes al Finis Terrae II, el nodo de login asignado es diferente al cual en el que habías configurado el cron te encontrarás que este no contiene las tareas que esperabas, lo que te puede llevar a pensar que se perdió tu configuración y volver a añadirlas en este nuevo nodo de login y por lo tanto, esas tareas se estarían ejecutando dos veces en diferentes nodos de login. Para evitar este tipo de situaciones o que tengas que acordarte en qué nodo de login habías configurado el cron y que tengas que acceder a él cada vez que quieras hacer alguna gestión en el cron, hemos preparado los nodos de login para que todos ellos puedan gestionar el mismo cron como si se tratase de un único nodo. De este modo, todos los nodos de login tendrán una copia de tu cron pero solo se ejecutará en uno de ellos.

Otra ventaja del cron redundante, es que aunque se caiga uno de los nodos frontales, tu cron se ejecutará sin problemas, mientras que si no se usa y el nodo que no funciona es en el que tenías establecido el cron, las tareas ahí configuradas no se ejecutarán con los consiguientes problemas.

Para utilizar el cron redundante deberás seguir los siguientes pasos:

1. Añadir las siguientes líneas al inicio del cron:

SHELL=/bin/bash
PATH=/sbin:/usr/sbin:/usr/local/sbin:/bin:/usr/bin:/usr/local/bin:

Nota.- En el caso de que ya tengas definida la variable PATH en tu cron, deberás asegurarte de que esté incluido el directorio ***/usr/local/bin*** *en dicha variable.*

2. Añadir "***run***" delante de cada comando a ejecutar en el cron. Por ejemplo, si tenemos una línea como esta:

15 12 * * * comando

Tendrías que modificarla para que quedase así:

15 12 * * * run comando

3. Tras haber editado y guardado el cron, se deberá ejecutar el comando "***sync_cron***" para sincronizar este nuevo cron con todos los nodos de login.

Nota.- Es muy importante acordarse de ejecutar este comando cada vez que se haga un cambio en el cron pues de no hacerlo, se podrían perder los cambios hechos o que no se ejecuten las tareas indicadas porque el nodo que tenga que hacerlo no tenga la configuración adecuada.

Nota.- Se debe tener en cuenta, que estas tareas serán ejecutadas en un nodo de login, por lo que no deberán ser tareas costosas en cuanto al consumo de recursos.