CAGE – Grupo de Investigación

SC3 – Super Cómputo y Cálculo Científico – UIS

@sc3uis

Introducción a SLURM

Gestión de trabajos en clusteres HPC

Módulo 1 – Semillero CAGE

Luis Alejandro Torres Niño | sc3.operaciones@uis.edu.co

Objetivos del módulo

- ➔ Entender **qué** es un gestor de recursos/colas y **por qué** es indispensable en un cluster compartido.
- ➔ Conocer la arquitectura de **Guane-1** (o el cluster del SC3 vigente): particiones, nodos, colas.
- ➔ Dominar los comandos esenciales: `sinfo`, `squeue`, `srun`, `salloc`, `sbatch`, `scancel`.
- ➔ Escribir **scripts de trabajo (batch jobs)** correctos y reproducibles.
- ➔ Usar **módulos de ambiente** (`module`) para cargar software científico.
- ➔ Resolver retos de programación individuales y un reto grupal de cierre.

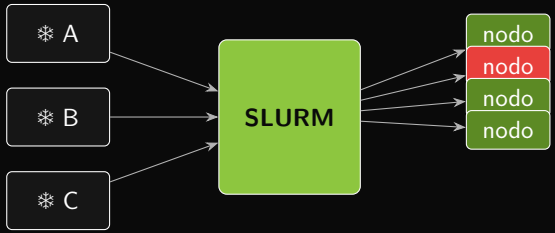
Agenda

1. ¿Qué es un gestor de trabajos?
2. Arquitectura de un cluster HPC y particiones
3. Conexión y entorno de trabajo
4. Comandos de consulta: `sinfo`, `squeue`
5. Ejecución interactiva: `srun`, `salloc`
6. Módulos de ambiente (`module`)
7. Scripts batch y `sbatch`
8. **Laboratorio**: retos individuales
9. `scancel` y buenas prácticas
10. **Reto grupal** de cierre

01
¿Qué es un gestor de trabajos?

El problema: recursos compartidos

- Un cluster HPC tiene **decenas o cientos** de nodos de cómputo, pero **muchos usuarios**.
- Sin control, dos personas podrían lanzar procesos en el mismo nodo y **competir por CPU/memoria**.
- Se necesita un **árbitro** que:
  - Reciba solicitudes de recursos.
  - Decida **cuándo** y **dónde** ejecutarlas.
  - Garantice aislamiento y **equidad**.



¿Qué es SLURM?

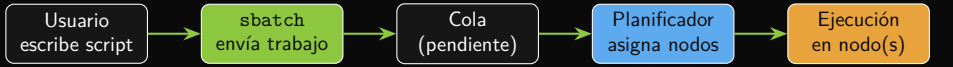
SLURM – Simple Linux Utility for Resource Management

Software de **código abierto** para administración de trabajos y gestión de clusteres Linux. Es el gestor de colas más usado hoy en el TOP500.

Sus cuatro responsabilidades principales:

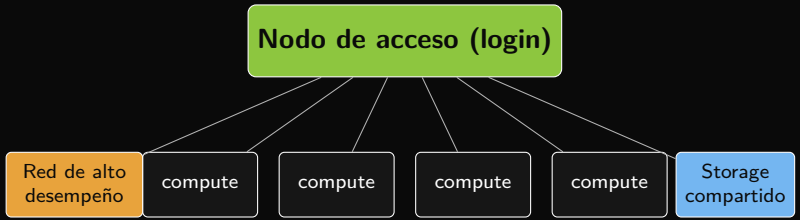
1. **Monitorea** los recursos del cluster (qué nodos existen, su estado, su carga).
2. **Recopila** las solicitudes de recursos de los usuarios (trabajos).
3. **Asigna prioridad** a los trabajos en cola según política (FIFO, *fair-share*, etc.).
4. **Ejecuta** los trabajos en los nodos de cómputo asignados y reporta resultados.

Flujo de trabajo con SLURM



02
Arquitectura del cluster y particiones

Anatomía de un cluster HPC



El usuario **nunca** ejecuta cargas pesadas en el nodo de acceso: solo edita, compila liviano y **envía trabajos** a los nodos de cómputo vía SLURM.

Particiones

Los nodos de cómputo se agrupan en **particiones** (conjuntos lógicos) según su hardware o función.

Partición	Uso típico
`legacy*`	partición por defecto, CPU
`gpu_titan`	nodos con GPU NVIDIA Titan
`fat`	nodo de memoria alta (*fat node*)
`amd`	nodos GPU/CPU AMD (ROCm)

★ Tip

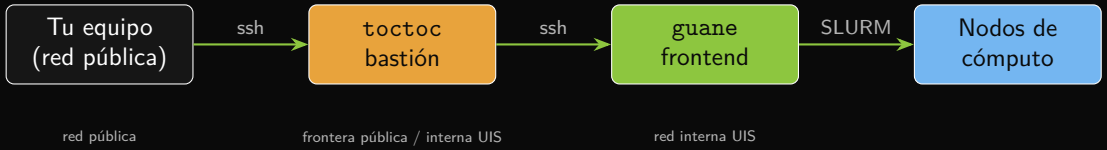
El asterisco `*` junto al nombre indica la partición **por defecto** si no se especifica ninguna.

❄ Cuidado

Pedir la partición equivocada (ej. `gpu_titan` sin usar GPU) desperdicia recursos escasos para otros usuarios del grupo.

Arquitectura de red: el bastión `toctoc`

- El cluster **no es accesible directamente** desde internet: vive en la red interna de la UIS.
- Para entrar, primero se pasa por un **nodo bastión** (*jump host*): `toctoc.sc3.uis.edu.co`.
- El bastión es el único punto expuesto a la red pública; desde ahí se salta a `guane` (frontend del cluster).



Conexión al cluster (dos saltos)

```
# 1. Desde tu equipo, conectate al bastion
ssh user_name@toctoc.sc3.uis.edu.co

# 2. Una vez DENTRO de toctoc, salta al frontend del cluster
[user_name@toctoc ~]$ ssh guane
[user_name@guane ~]$   # ya estas en el frontend, listo para usar SLURM
```

❄ Cuidado

Los dos saltos son **sesiones SSH independientes**: si cierras la conexión a `toctoc`, pierdes también la sesión en guane.

Acceso directo con ProxyJump

Configura tu `~/.ssh/config` **local** para llegar a guane en un solo comando, sin pasar manualmente por toctoc:

```
Host toctoc
    HostName toctoc.sc3.uis.edu.co
    User tu_usuario

Host guane
    HostName guane
    User tu_usuario
    ProxyJump toctoc
```

★ Resultado

Con esta configuración, `ssh guane` desde tu máquina local abre automáticamente el túnel a través de toctoc y te deja directo en el frontend del cluster.

03
Comandos esenciales de consulta

`sinfo` – estado del cluster

- ➔ Muestra la información de los nodos y de las particiones.
- ➔ Un asterisco (*) tras el nombre de partición indica que es la **partición por defecto**.
- ➔ Un asterisco tras el estado del nodo indica que **no está respondiendo**.

```
[user_name@guane ~]$ sinfo

PARTITION   AVAIL  TIMELIMIT  NODES  STATE  NODELIST
legacy*     up     infinite   8      idle   guane[01-08]
gpu_titan   up     infinite   1      down*  yaje
gpu_titan   up     infinite   1      idle   felix
fat         up     infinite   1      idle   thor
amd         up     infinite   2      mix    exadell,smexa
```

Estados de nodo

`idle`
libre

`mix`
parcialmente ocupado

`alloc`
completamente ocupado

f

★ ¿Por qué importa?

Antes de pedir recursos con `--partition`, revisa con `sinfo` si hay nodos `idle` disponibles: te ayuda a estimar cuánto esperarás en la cola.

squeue – cola de trabajos

Muestra los trabajos en cola de todo el cluster o de un usuario específico.

```
[user_name@guane ~]$ squeue -u $USER

JOBID   PARTITION   NAME         USER       ST   TIME      NODES
    NODELIST(REASON)
18276   gpu_titan   entrenar_gpu user_name  R    7:46:29   1     felix
18277   legacy      analisis_04  user_name  R    7:39:41   1     guane02
18282   amd         simulacion   user_name  R    2:33:47   1     exadell
```

Estados (columna ST):

- ➔ `R` = Running (ejecutando)
- ➔ `PD` = Pending (en espera)
- ➔ `CA` = Cancelled (cancelado)

■ Mini-reto

Ejecuta `squeue` (sin `-u`) y cuenta cuántos trabajos hay actualmente en la partición `legacy`.

04
Ejecución interactiva

srun – ejecución directa

srun permite ejecutar una aplicación de forma directa con las opciones determinadas por el usuario.

```
[user_name@guane ~]$ srun --ntasks=4 --partition=legacy --label /bin/
    hostname
2: guane01.uis.edu.co
1: guane01.uis.edu.co
0: guane01.uis.edu.co
3: guane01.uis.edu.co

# Forma corta equivalente
[user_name@guane ~]$ srun -n 4 -p legacy -l /bin/hostname
```

❄ Cuidado

srun **bloquea tu terminal** hasta que el trabajo termina. Para tareas largas usa sbatch (más adelante).

`salloc` – reserva interactiva

- ➔ Obtiene la asignación de un trabajo con **acceso a consola**.
- ➔ Los recursos reservados son los especificados en *options*.
- ➔ Permite trabajar de forma interactiva dentro de un nodo de cómputo (ideal para depurar).

```
[user_name@guane ~]$ salloc --nodes=1 --ntasks=1 --partition=legacy --
    time=00:30:00
salloc: Granted job allocation 18301
[user_name@guane ~]$ srun --pty /bin/bash
[user_name@guane03 ~]$   # ahora estas DENTRO del nodo de computo
```

■ Reto individual

Conéctate con `salloc` a un nodo de la partición `legacy`, ejecuta `hostname` y `nproc` dentro de él, y sal con `exit` **dos veces** (shell del nodo y reserva de `salloc`) para liberar los recursos.

05
Módulos de ambiente

¿Qué son los módulos de ambiente?

- Un **módulo** empaqueta variables de entorno (`PATH`, `LD_LIBRARY_PATH`, etc.) dentro de un script.
- Se define un módulo por aplicación y establece el ambiente apropiado para su ejecución.
- Permiten tener **múltiples versiones** del mismo software (ej. dos versiones de GCC) sin que choquen entre sí.

Comandos principales	
`module avail`	lista todos los módulos disponibles
`module load NOMBRE`	carga las variables de un módulo
`module list`	lista los módulos cargados actualmente
`module unload NOMBRE`	remueve un módulo específico
`module purge`	elimina todos los módulos cargados

Usando módulos

```
[user_name@guane ~]$ module av
-------------------- /opt/ohpc/pub/moduledeps/gnu14 --------------------
  mpich-ucx/5.0.1   mpich/5.0.1   openmpi5/5.0.7 (L)

----------------------- /opt/ohpc/pub/modulefiles -----------------------
  cuda/11.8            gromacs/legacy   gromacs/titan (D)   openfoam/amd
  gnu14/14.2.0 (L)     gromacs/amd      lammps/legacy        openfoam/legacy
  gnu15/15.2.0         lammps/titan (D) lammps/amd           openfoam/titan (D)
  rocm/system          orca/6.1.1       cp2k/2026.1-amd       ...

[user_name@guane ~]$ module load openfoam/legacy
[user_name@guane ~]$ module list
Currently Loaded Modulefiles:
  1) gnu14/14.2.0  2) openmpi5/5.0.7  3) openfoam/legacy

[user_name@guane ~]$ module unload openfoam/legacy
[user_name@guane ~]$ module purge
```

★ Tip – convención de nombres

Mucho software está compilado **por partición/arquitectura**: `openfoam/legacy` corre en CPU, `openfoam/titan` usa GPU NVIDIA, `openfoam/amd` usa GPU AMD. `(D)` = versión por defecto, `(L)` = módulo ya cargado.

Explorando módulos disponibles

Cuando el listado de `module avail` es muy largo, existen atajos:

```
# Solo los modulos por defecto (uno por software)
[user_name@guane ~]$ module --default avail    # o: ml -d av

# Cuenta cuantas versiones hay de cada software
[user_name@guane ~]$ module overview           # o: ml ov

# Busca en TODOS los modulos, incluso los no visibles aun
[user_name@guane ~]$ module spider gromacs

# Busca por palabra clave en nombre y descripcion
[user_name@guane ~]$ module keyword gpu mpi
```

Mini-reto

Usa `module spider quantum-espresso` y determina qué versión deberías cargar para correr en la partición `amd`.

06
Scripts batch: `sbatch`

Anatomía de un script SLURM

```
#!/bin/bash
# Solicitud de recursos
#SBATCH --partition=legacy
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=1G

# Tiempo de ejecucion
#SBATCH --time=1-12:30:00

# Nombre y salidas
#SBATCH --job-name=myjob
#SBATCH --output=myjob.out
#SBATCH --error=myjob.err

# Carga de modulo
module load openfoam/legacy

blockMesh
```

Tres bloques:

1. **Preliminares**: intérprete (`#!/bin/bash`), siempre primera línea.
2. **Directivas SLURM**: empiezan con `#SBATCH`; bash las ignora, SLURM las interpreta. Van antes de cargar módulos.
3. **Comandos**: carga de módulos + ejecutable/script real.

❄ Error común

Un espacio entre `#` y `SBATCH` hace que la directiva sea ignorada silenciosamente.

Directivas SBATCH más usadas

Directiva	Significado
`--job-name=NAME`	Nombre del trabajo (visible en `squeue`)
`--partition=P`	Partición donde se ejecuta
`--nodes=N`	Número de nodos solicitados
`--ntasks=N`	Número total de tareas (procesos)
`--ntasks-per-node=N`	Tareas por nodo
`--cpus-per-task=N`	Hilos/cores por tarea (para OpenMP)
`--mem=SIZE`	Memoria por nodo (ej. `4G`)
`--time=D-HH:MM:SS`	Tiempo límite de ejecución
`--output=FILE`	Archivo de salida estándar
`--error=FILE`	Archivo de salida de error
`--array=1-N`	Lanza N trabajos indexados (array job)

Enviar y monitorear un trabajo

```
[user_name@guane ~]$ sbatch myjob.slurm
Submitted batch job 18290

[user_name@guane ~]$ squeue -u $USER
JOBID   PARTITION   NAME    USER       ST   TIME     NODES NODELIST(
    REASON)
18290   legacy      myjob   user_name  R    0:05     1     guane05

[user_name@guane ~]$ cat myjob.out
... salida estandar del programa ...
```

- ➔ `sbatch` envía el *batch_file* a SLURM para su ejecución **en segundo plano**.
- ➔ Si el envío es exitoso, SLURM retorna el `JOBID` del trabajo.
- ➔ Puedes cerrar tu sesión SSH: el trabajo sigue corriendo en el cluster.

Laboratorio: retos individuales

Reto 1 – Tu primer batch job

Individual

1. Escribe un script `hola.slurm` que imprima `"Hola desde $(hostname)"` y la fecha (`date`).
2. Solicita 1 nodo, 1 tarea, partición `legacy`, tiempo límite de 5 minutos.
3. Envíalo con `sbatch` y verifica su estado con `squeue`.
4. Revisa el contenido de `slurm-<jobid>.out` (o el nombre que configuraste con `--output`).

Reto 2 – `sinfo` y particiones

■ Individual

1. Con `sinfo`, identifica cuántos nodos `idle` hay en `legacy` ahora mismo.
2. Modifica tu script del Reto 1 para pedir `amd` en vez de `legacy`.
3. Compara los tiempos: ¿cuánto tarda en pasar de `PD` a `R` en cada partición?
4. Agrega `--cpus-per-task=4` y dentro del script imprime `nproc`. ¿El número coincide con lo solicitado?

Reto 3 – Array jobs

■ Individual

Un *array job* lanza *N* copias del mismo script, cada una con un índice distinto en la variable `$SLURM_ARRAY_TASK_ID`.

```
#!/bin/bash
#SBATCH --partition=legacy
#SBATCH --array=1-5
#SBATCH --output=tarea_%a.out

echo "Soy la tarea numero $SLURM_ARRAY_TASK_ID"
sleep $((SLURM_ARRAY_TASK_ID * 2))
echo "Tarea $SLURM_ARRAY_TASK_ID terminada"
```

1. Envía este script y observa con `squeue` cómo aparecen las 5 subtareas.
2. Revisa los 5 archivos `tarea_*.out` generados.

07

`scancel` y buenas prácticas

`scancel` – cancelar trabajos

- Envía una señal al *job* y/o a sus subprocesos.
- Por defecto envía `SIGTERM` (y `SIGKILL` si el trabajo no termina); con `--signal` eliges otra señal.
- El trabajo cancelado es el que corresponde al `jobid` (obtenido con `squeue`).

```
[user_name@guane ~]$ scancel 18290

# Se pueden usar filtros:
[user_name@guane ~]$ scancel --user=$USER
[user_name@guane ~]$ scancel --user=$USER --partition=gpu_titan --state=
    PENDING
```

Buenas prácticas en un cluster compartido

- ➔ **Nunca** ejecutes cargas pesadas directamente en el nodo de acceso: siempre usa `srun`/`sbatch`.
- ➔ Pide **solo los recursos que necesitas**: sobre-pedir memoria/cores bloquea el acceso a otros usuarios.
- ➔ Define siempre `--time`: un límite realista libera el nodo si tu trabajo se cuelga.
- ➔ Usa `--job-name` descriptivo y `--output`/`--error` separados para depurar más fácil.
- ➔ Borra o archiva tus salidas viejas: el *storage* compartido también es un recurso finito.
- ➔ Antes de pedir ayuda, revisa `myjob.err` – la mayoría de errores se explican solos ahí.

Reto grupal de cierre

Reto grupal – Pipeline de 3 etapas

★ Grupos de 2-3 personas

Construyan un **pipeline con dependencias** usando 3 scripts SLURM encadenados:

1. `etapa1.slurm`: genera un archivo `datos.txt` con 100 números aleatorios (uno por línea, usa `$RANDOM` en bash).
2. `etapa2.slurm`: lee `datos.txt` y escribe en `stats.txt` el mínimo, máximo y promedio (usa `awk` o un script en C que programarán ustedes mismos).
3. `etapa3.slurm`: imprime `stats.txt` con un mensaje de `"Pipeline completado por el grupo <nombre>"`.

Requisito clave: etapa2 no debe iniciar hasta que etapa1 termine, ni etapa3 hasta que etapa2 termine. Investiguen y usen:

```
1 sbatch --dependency=afterok:<jobid> etapa2.slurm
```

Reto grupal – Entregables y evaluación

Al final de la sesión, cada grupo debe mostrar:

- ➔ Los 3 scripts `.slurm` funcionando encadenados con `--dependency`.
- ➔ Captura de `squeue` mostrando las 3 etapas (una `PD` esperando dependencia, mientras otra corre).
- ➔ El archivo `stats.txt` final con resultados correctos.
- ➔ **Bono**: paralelizar la etapa 1 como *array job* (cada tarea genera un archivo distinto) y que la etapa 2 los consolide.

Cierre

★ Lo esencial de SLURM

Pide **solo los recursos que necesitas**, prueba en pequeño con `salloc`, y automatiza con `sbatch`, *arrays* y dependencias.

¡Gracias!

sc3.operaciones@uis.edu.co