Formación Slurm

INSTITUT BOTÀNIC DE BARCELONA
Anima ITS training services
Agosto 2025

Introducción

Plataformas HPC

- Versatilidad
  - Cluster de cómputo (CPU / GPU / Aceleradores...)
  - Cluster almacenamiento
  - Red altas prestaciones
  - ...
- Escalabilidad
  - capacidad
  - rendimiento
  - partiendo de 1 nodo

Plataforma > servicios

Estructura modular acorde a dimensión y necesidades

- Sistema gestión recursos y trabajos
- Sistema de ficheros paralelo
- Servicio directorio (integrado o externo)
- Servicios acceso remoto mejorado
- Servicios visualización altas prestaciones
- Catálogo software centralizado
- ...

Anima > servicios

- Traspaso conocimiento modular (a medida)
- Servicio adaptación
- Servicio mantenimiento

AUTORIZACIÓN Y AUTENTICACIÓN (AA)

AA > usuarios y grupos

AA > Linux

Identificación

- Linux: `UID` / `GID` (`/etc/passwd`, `/etc/group`)
- Clasificación (implícita) de cuentas:
  - sistema (UID/GID < 1000)
  - usuario (UID/GID >= 1000)

Autenticación

- password
- ssh-keys

Autorización

- permisos de fichero (clásicos, ACL)

Directorio de inicio (home)

En Linux, suele estar dentro de `/home`

En el cluster, /home es relativamente pequeño comparado con el almacenamiento paralelo

 Usar /home solo para administradores, y el sistema de ficheros paralelo para los home de los usuarios

AA > comandos linux

GNU libc:

- getent

GNU coreutils:

- id
- groups
- whoami
- who

AA > ejemplos identificación de usuarios

```
# query -> getent (based on NSS)
# getent DB KEY
getent passwd afont
getent group quimica
# effective uid/gid -> id
id afont
# nombre -> uid
id -nu afont
```

AA > Creación de cuentas Linux

Herramientas estándar para gestión de usuarios y grupos (`adduser`, `usermod`...)

Ejemplo creación usuario

```
# en master
useradd -u UID -g GID [-m] -d HOME_DIR USERNAME
# añadir usuario a grupo
usermod -aG GRUPO USERNAME

# comprobar
su - USERNAME
```

AA > Autenticación

Autenticación básica > contraseñas

En Linux, las contraseñas se guardan (cifradas) en `/etc/shadow`

Hoy en día, las contraseñas tienen sentido sobre todo para autenticación local (directamente por consola, no por SSH)

(accediendo por SSH, □ pares de llaves SSH)

AA > Cambio contraseña Linux

```
# individual
passwd

# en lotes
chpasswd < lote_usuarios.pass
```

Formato chpasswd:

```
user_name:password
```

AA > Autorización

En UNIX y derivados, "todo son ficheros"

Mediante los atributos de los ficheros controlo acceso a recursos

Atributos clásicos:

- pertenencia (IDs de usuario y grupo)
- modo de acceso
  - lectura (r), escritura(w), ejecución (x)
  - usuario (u), grupo (g), otros (o)

Comandos: `chown`, `chgrp`, `chmod`

Autorización clásica/ejemplos

```
chown afont RUTA
chgrp ml RUTA
chmod u+rw RUTA
chmod o-rwx RUTA
```

Reto: membresía compleja

¿Cómo restrinjo el acceso a 3 usuarios que pertenecen a grupos diferentes?

ACL

Access Control Lists

Más flexibles (no constreñidas a 1 usuario + 1 grupo)

(El soporte depende del sistema de ficheros en particular)

ACL/ejemplos

Entrada = `TIPO:NOMBRE:PERMISOS`

```
getfacl RUTA

# u -> usuario; g -> grupo
# -R -> recursivo
setfacl -m "u:afont:rw" RUTA
setfacl -m "g:ml1:r" RUTA

# quitar
setfacl -x "u:laura" RUTA
```

filesystem > quotas

Limitar a usuarios / grupos el espacio en disco que pueden utilizar

Dependiendo del sistema de ficheros en concreto, el comando puede variar

quotas > ext4

Comandos:

- repquota: mostrar información
- edquota: modificar las cuotas
  - -u / -g : usuario o grupo
  - -f : qué sistema de ficheros

quotas > ext4 > ejemplos

```
# activar cuotas para usuarios y grupos en /home
# (si no estaban activadas ya)
quotaon -ugv /home

# informe: usuarios en todos los sistemas de ficheros
repquota -a -u
# informe: grupo solo /home
repquota -g /home

# modificar cuota usuario jdoe
edquota -u jdoe
```

xfs > xfs_{quota}

```
xfs_quota -c 'COMANDO' SISTEMA_FICHEROS
```

Opciones generales:

- -x: expert mode (comandos de administrador: report, limit, ...)

Comandos:

- quota
- limit

xfs_{quota} > quota

Muestra utilización y límites para un usuario o grupo (nombre o ID)

- `quota [ -g | -u ] [ -hnNv ] [ -f file ] [ ID | name ]`
  - -h human-readable
  - -n muestra ID en vez de nombre
  - -N omite la cabecera
  - -v verbose
  - -f guardar en FILE

xfs_{quota} > limit

Define la cuota: estricta (hard) o cautelar (soft), tanto para el tamaño (bloques, b) como para los inodos (inode, i)

- `limit [ -u | -g ] bsoft=N | bhard=N | isoft=N | ihard=N [ -d ] ID | NAME`
  - `-d`: valor por defecto
  - -u usuario, -g grupo

xfs > xfs_{quota} > ejemplos información

```
xfs_quota -c 'quota -hu USER' /home
xfs_quota -c 'quota -hg GROUP' /home

# INFORME
xfs_quota -x -c 'report -h' /home
```

xfs > xfs_{quota} > ejemplos cuotas

```
# USUARIO
# limitar en /home para usuario USER el disco a 512MB cautelar, estricto 1GB
xfs_quota -x -c 'limit bsoft=512m bhard=1g USER' /home
# limitar inodos: 500 cautelar, 700 estricto
xfs_quota -x -c 'limit isoft=500 ihard=700 USER' /home
# quitar limites de disco (bloques)
xfs_quota -x -c 'limit bsoft=0 bhard=0 USER' /home

# GRUPO
xfs_quota -x -c 'limit -g bsoft=1000m bhard=1200m GROUP'
```

—

HPC

Escenario típico en clusters

- recursos diversos en varios nodos
- múltiples usuarios x múltiples trabajos

 Conocer el estado de los recursos (observación)

 Automatizar el buen uso de los recursos (planificación)

 Gestionar el desbordamiento de los recursos (colas)

Este escenario se plantea en las diversas manifestaciones de clusters (HPC, kubernetes, big data...)

Al sistema que intermedia se le suele conocer como "gestor de cluster", "planificador", "sistema de colas", ...

Slurm > introducción

Presentación Slurm

```
Slurm is an...
  open source,
  fault-tolerant, and
  highly scalable
cluster management and job scheduling system
  for large and small Linux clusters.
```

Slurm/Functions

```
As a cluster workload manager, Slurm has three key functions:
```

1. `it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work.`
2. `it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes.`
3. `it arbitrates contention for resources by managing a queue of pending work.`

Slurm/Características

Versátil

Sintaxis clara

Código abierto

Desarrollo activo

Plataformas:

- PC / Linux
- IBM BlueGene
- IBM Parallel Environment
- Cray/XT

Slurm > Historia y versiones

2002: LLNL Slurm (solo gestor recursos)

2006: Slurm 1.0.0.1

2007: Slurm 1.2.0.1

2010: SchedMD, Slurm 1.3.1 (planificacion trabajos)

2014: Slurm 14-03-1-1

2024: Slurm 24.11

2025: Slurm 25.5

Slurm > conceptos

- Recursos: CPU, RAM, TRES, GRES
- Nodos, Clusters
- Particiones ("colas")
- Pasos, Trabajos

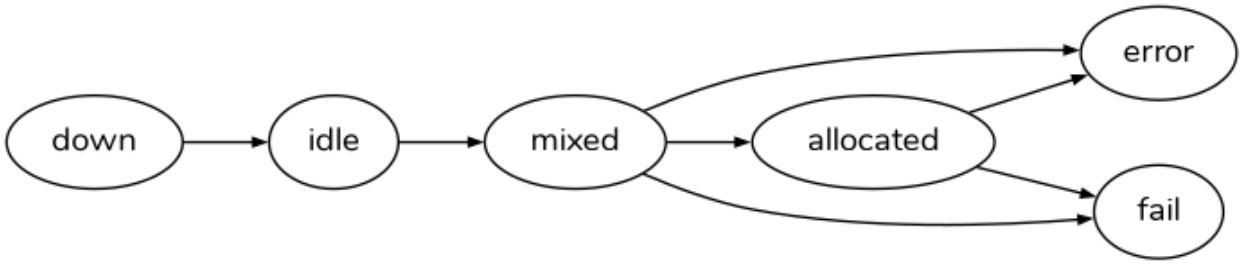
Conceptos > TRES y GRES

TRES Trackable RESource. Recurso que puede limitarse o monitorizarse

- CPU
- Mem
- GRES
- ...

GRES Generic RESource. Recursos definidos localmente (típicamente, aceleradores)

Ciclos de vida > nodos > estados



Ciclos de vida > nodos > flags

Flag	Significado
Drain	Drenado (no acepta nuevos trabajos)
Completing	Trabajos en estado "Completing"
Reserved	Solo disponible para la reserva
$\text{powering}_{\text{up}}$	Arrancando
$\text{powering}_{\text{down}}$	Apagándose
$\text{powered}_{\text{down}}$	Apagado

Tipologías de trabajo

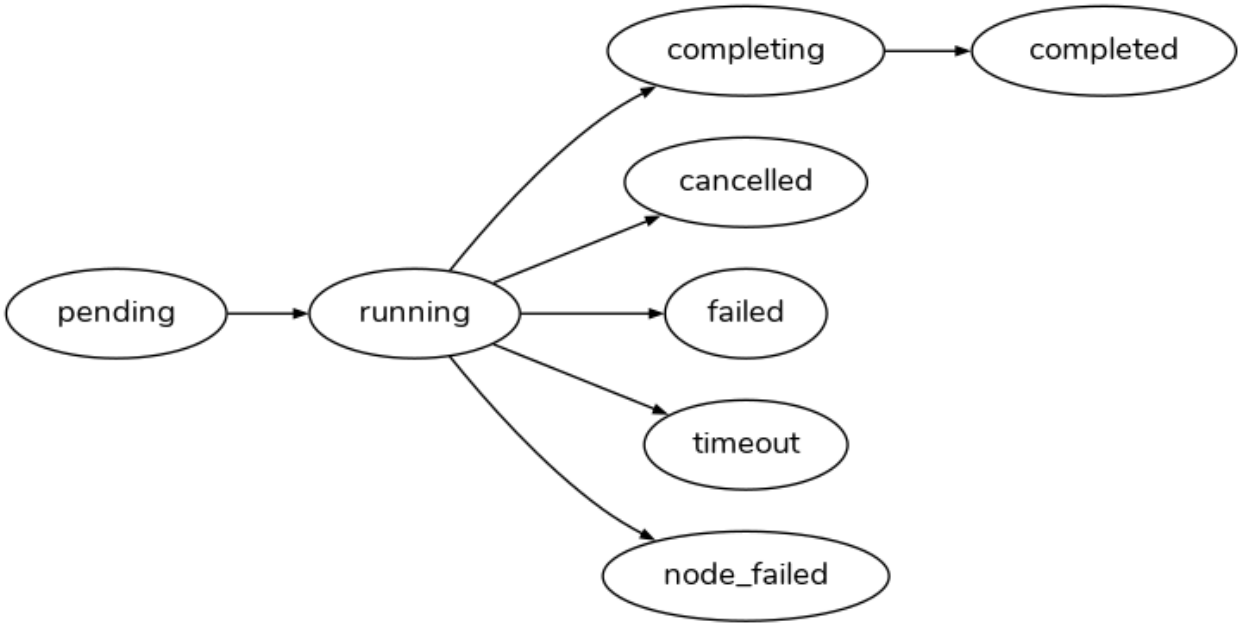
Interactividad

- Uso interactivo
- Proceso "por lotes" (batch)

Paralelismo

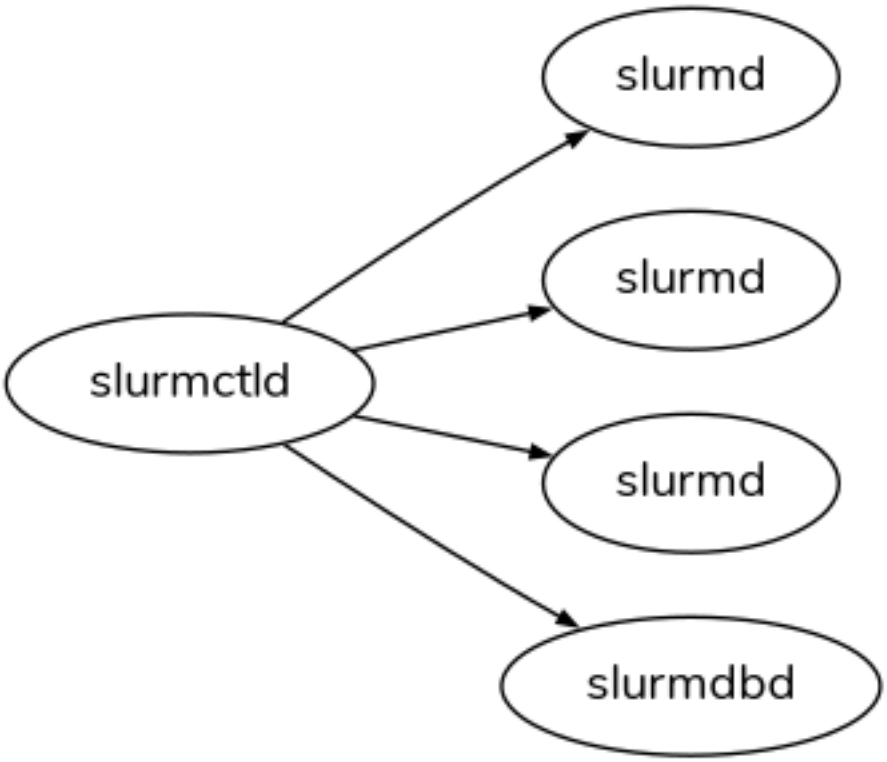
- secuencial / serie
- paralelo

Ciclos de vida > trabajos



Slurm > arquitectura

Servicios



Munge

`MUNGE (MUNGE Uid 'N' Gid Emporium) is an authentication service for creating and validating credentials. It is designed to be highly scalable for use in an HPC cluster environment.`

Una forma sencilla de ”fiarse” de los servicios distribuidos entre controlador y nodos.

 UIDs / GIDs comunes

`/etc/munge/munge.key`

USO DE SLURM

Flujos de trabajo

Niveles

- Proceso
- Paso
- Trabajo
- Conjunto de trabajos

Comandos Slurm

Slurm	PBS
srun	-
sbatch	qsub
squeue	qstat
scancel	qdel
sinfo	qstat -a
scontrol hold	qhold
scontrol release	qrls

Uso slurm > ejecución

srun

Ejecución de un ”paso”

Parámetros:

```
--ntasks=1
--tasks-per-node=
--cpus-per-task=
# por nodo
--gres=gpu:hopper:2

--mem=<size><unit>
--mem-per-cpu=<size><unit>
```

indicar al menos ntasks

srun > sesiones interactivas

En cada caso, ajusto los recursos que reservo (ver referencia comando srun)

```
# sesión interactiva en consola (sin ventanas)

# Ejemplo 1: reservo 2 CPU
srun --cpus-per-task=2 --pty -p gpu bash

# Ejemplo 2: GPU. Solicito cola gpu (-p gpu), 2 GPU y 4 CPU
srun --cpus-per-task=4 --gres=gpu:2 --pty -p gpu bash
```

srun > Aplicaciones gráficas

¿Dónde?

- Nodo visualización
- Nodo login
- Nodo cálculo

Importante: para que srun pueda abrir ventanas, las ventanas ya tienen que funcionar en nuestra conexión al nodo de login...

- cliente X11 en nuestro PC
- `ssh -Y` (o equivalente)

```
# sersión interactiva con ventanas
srun --pty -n 1 --x11 bash
# una vez en el nodo, puedo ejecutar la app gráfica, por ejemplo...
xterm
```

Trabajos/scripts Slurm

Fichero de descripción de trabajos (bash, python...)

Parámetros dentro de fichero / en la línea de comandos

`job.sbs`

```
#!/bin/bash
#SBATCH --time=00:01:00
srun hostname
```

Ejemplo envío:

```
sbatch --time=00:30:00 job.sbs
```

sbatch/Parámetros ejecución

```
--partition=

--ntasks=
--tasks-per-node=
--cpus-per-task=

--exclusive

# por nodo
# cualquier GPU
--gres=gpu:1
# un tipo de GPU en particular
--gres=gpu:hopper:1
```

```
--mem-per-cpu=<size><unit>
# por nodo
--mem=<size><unit>
```

Indicar memoria (`mem` o `mem-per-cpu`)

sbatch/Parámetros contexto

```
# DD-HH / HH:MM:SS
--time=
# YYYY-MM-DDTHH:MM:SS
--begin=
# YYYY-MM-DDTHH:MM:SS
--deadline

--workdir=
--output=
--error=
--export
--job-name=
```

Indicar `time`, aunque sea "a ojo"

Avisos por e-mail

MailProg=/usr/bin/mail

Parámetros sbatch:

```
# NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalente a los anteriores),
# TIME_LIMIT, TIME_LIMIT_90 (90% limite), TIME_LIMIT_80, TIME_LIMIT_50,
# ARRAY_TASKS (un correo por tarea)
--mail-type=END
# dirección de email
--mail-user=j@superduper.hpc
```

En mail-type se pueden incluir varias opciones separadas por coma.

sbatch/Dependencias entre trabajos

```
--dependency=after:JOB_ID
--dependency=afterok:JOB_ID
--dependency=afternotok:JOB_ID
--dependency=singleton
```

sbatch/Variables

```
${SLURM_JOB_ID}
${SLURM_TASK_PID}
${SLURM_JOB_NAME}
${SLURM_LOCALID}
${SLURM_JOB_NODELIST}
${SLURM_JOB_PARTITION}
${SLURM_SUBMIT_DIR}
${SLURM_SUBMIT_HOST}
```

```
${SLURM_NTASKS}
${SLURM_TASKS_PER_NODE}
${SLURM_CPUS_PER_TASK}
${SLURM_GPUS}
...
```

scratch

En ocasiones, puede resultar más práctico recurrir al disco local de los nodos ("scratch")

Indicar al programa dónde escribir (parámetros, variables de entorno...)

Variables

- `TMPDIR=/scratch/${SLURM_JOB_ID}`

La creación de directorios temporales en /scratch se puede realizar...

- en el script .sbs
- con un prolog script

scratch/ciclo de vida

- crear directorio temporal en scratch
- copiar los ficheros de entrada (dependiendo del programa)
- procesos escriben en scratch
- llevar al almacenamiento compartido los ficheros importantes
- eliminar el directorio temporal

USO SLURM > GESTIÓN

Campos informativos

- squeue
- sacct
- sinfo

Por nombre de campo o nombre abreviado (abrev.)

El código abreviado permite indicar el ancho del campo y la alineación (por defecto a la izquierda; a la derecha si se incluye punto).

Ejemplos: `%.9i`, `%8M`

campos informativos (I)

Campo	Abrev.	Squeue	Sacct
job id	%i	jobid	
Cola	%P	partition	
Nombre Trabajo	%j		
	%u	user	
Momento envio	-	SubmitTime	Submit
Nombres nodos	%N		
CPU (Available/Idle/Other/Total)	%C		
Memoria disponible	%e		

Campo	Abrev.	Squeue	Sacct
memoria por nodo	%m		

campos informativos (II)

Campo	Abrev.	Squeue	Sacct
Carga CPU	%O		
GRES	%G		
Tiempo empleado	%M		
Nodos asignados	%N		
Motivo del estado	%r	reason	

squeue

Partición ~ cola

```
squeue
```

```
squeue -u USER
```

```
# Campos abreviados: -o
squeue -o "%6i %9P %20j %10u %8M %10r %N"
JOBID  PARTITION  NAME    USER    TIME     REASON     NODELIST

squeue -o "%.9i %.9P %.16j %.8u %10Q %.2t %.10M %.6D %.4C"
JOBID  PARTITION  NAME    USER    PRIORITY   ST       TIME  NODES CPUS
```

squeue > ejemplos (II)

```
# Campos con nombre completo: -O
squeue -O "JobId,SubmitTime"

# squeue -t STATE -p PARTITION
```

STATE CAncelled, CompleteD, CompletinG, DeadLine, Failed, NodeFailed, OutOfMemory, PenDing (PD), Running, ReQueued, STopped, Suspended, TimeOut,...

sinfo

Información sobre nodos y colas

Parámetros

`--format` qué información mostrar (ver campos disponibles más adelante)
`-s` resumen
`-N` resumen por nodos
`-l` ("long") mostrar info adicional

sinfo > ejemplos

```
sinfo
```

```
sinfo -s
# ojo que por defecto el ancho de columna para
#   mem puede ser pequeño para fat nodes ;-)
sinfo -Nel

# Resumen por nodos
sinfo -N --format="%.8P %.8N %.16C %8e %8m %8O %G"

# resumen por colas
sinfo --format="%.9P %.16C %8m %8O %G"
```

scancel

```
scancel JOB_ID

# todos los trabajos pendientes
scancel --state=PENDING
```

Modelos de paralelización

Por concisión, en cada ejemplo se indican solo los parámetros que aplican al caso particular

Cabe recordar que se recomienda incluir en el script .sbs "todos" (aunque sea con estimaciones)

Trabajo secuencial

Un paso tras otro. Cada paso conlleva tarea única (`-n 1`)

```
#!/bin/bash

srun -n 1 comando_paso_1
srun -n 1 comando_paso_2
```

Trabajo paralelo (array)

Varios pasos simultáneos

```
#!/bin/bash
#SBATCH --array=0,4,8-12
echo "Job ID: ${SLURM_ARRAY_JOB_ID} Task ID: ${SLURM_ARRAY_TASK_ID} "
echo "Total: ${SLURM_ARRAY_TASK_COUNT}."
echo "Id min: ${SLURM_ARRAY_TASK_MIN}, max: ${SLURM_ARRAY_TASK_MAX}"

srun -n 1 comando ${SLURM_ARRAY_TASK_ID}
```

Slurm proporciona el índice (`${SLURM_ARRAY_TASK_ID}`) que luego yo debo facilitar a mi aplicación / comando.

Trabajo paralelo (script)

```
#!/bin/bash
#SBATCH --ntasks=2
#SBATCH --mem=512M
srun --exclusive -n 1 -N 1 -c 1 --mem 256M comando 1 &
srun --exclusive -n 1 -N 1 -c 1 --mem 256M comando 2 &
wait
```

Trabajo paralelo (multihilo)

```
#!/bin/bash
#SBATCH --cpus-per-task=4

# la sintaxis dependerá del programa, pero suele
#   ser -t NUM_HILOS
srun -n 1 --cpus-per-task=4 COMANDO_MULTIHILO \
 -t  ${SLURM_CPUS_PER_TASK}
```

Trabajo paralelo (MPI)

```
#!/bin/bash
# 2 nodos x 32 tareas / nodo == -n 64
#SBATCH --nodes=2
#SBATCH --tasks-per-node=32
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=1G

echo "Tareas: ${SLURM_NTASKS}"
echo "Tareas/nodo: ${SLURM_TASKS_PER_NODE} \
  CPUs/tarea: ${SLURM_CPUS_PER_TASK}"
srun programa_mpi
```

MPI sí es capaz de ejecutar automáticamente varios procesos de la misma app, por eso puedo usar n > 1

(concretamente, al omitir $-n$ en srun, utiliza el número de tareas de sbatch)

Configuración: slurm.conf

slurm.conf > categorías parámetros (I)

~500 parámetros

- descripción nodos
- gres
- descripción colas
- prioridades
- parámetros por defecto / límites básicos
- servicios
- ...

slurm.conf/categorías parámetros (II)

- control de acceso
- registro de eventos
- contabilidad
- registro de trabajos completados
- licencias
- "green computing"
- seguimiento estado nodos
- ...

protocolo configuración

- Replicar ficheros configuración en cada nodo
- reiniciar `slurmd` en nodos
- reiniciar `slurmctld` en plano de control: `systemctl restart slurmctld`

(para cambios simples, puede ser suficiente con `scontrol reconfigure`, en lugar de reiniciar los servicios)

slurm.conf > métodos de réplica

- NFS
  - añade una dependencia
  - en el export donde este la config puede haber otras configs que no convenga compartir en nodos de cómputo
- clustershell (`clush`): inmediata
- warewulf
  - gestionada
  - hay que actualizar el fichero en el overlay y reconstruir el overlay

Parámetros generales

```
SlurmctldHost=slurm(11.22.33.44)
# 0   manual; 2   auto
ReturnToService=2
# 0   disabled; 1    available
JobRequeue=0

# terminar paso (procesos) cuando cualquiera de ellos falla
KillonBadExit=1

MaxJobCount=10000
```

Nodos

```
NodeName=cpu[01-12] SocketsPerBoard=2 CoresPerSocket=10
    ThreadsPerCore=1 RealMemory=64000

NodeName=DEFAULT SocketsPerBoard=2 ThreadsPerCore=1 RealMemory=64000
  MemSpecLimit=4096

NodeName=cpu[01-12] CoresPerSocket=10

NodeName=gpu[14-18] SocketsPerBoard=2 CoresPerSocket=8
    ThreadsPerCore=1 RealMemory=64000 GRES=gpu:hopper:2
```

Incluir MemSpecLimit □

Acceso

Autorización a nivel de partición (cola)

- AllowUsers / DenyUsers
- AllowGroups / DenyGroups

Particiones/ejemplo

```
PartitionName=hpc Nodes=cpu[01-12] Default=YES
State=UP MaxNodes=4 AllowAccounts=graphene,visitors
```

Particiones > parámetros

```
AllocNodes
AllowAccounts, AllowGroups, AllowQos
DenyAccounts, DenyQos
Alternate
CpuBind
Default
DefCpuPerGPU, DefMemPerCPU, DefMemPerGPU, DefMemPerNode
MaxMemPerCPU, MaxMemPerNode, MaxNodes, MaxCPUsPerNode
MaxTime, MinNodes
DefaultTime
```

Particiones > parámetros (II)

```
DisableRootJobs
ExclusiveUser
GraceTime
Hidden
LLN
Nodes
OverSubscribe (Shared)
PartitionName
PreemptMode
PriorityJobFactor, PriorityTier
QOS
ReqResv
RootOnly
SelectTypeParameters
State
TRESBillingWeights
```

Slurm > scontrol

```
scontrol show node NODENAME

scontrol show job JOBID

scontrol show partition QUEUE

scontrol show config

scontrol reconfigure
```

scontrol/update job

```
# prolongar tiempo disponible para el trabajo
scontrol update JobID=JOBID TimeLimit+=2-0
```

```
scontrol update JobID=JOBID TimeLimit=10-0

# Estado     IDLE, DRAIN, FAIL, RESUME
#   FUTURE, DOWN, UNDRAIN
scontrol update NodeName=c03 State=ESTADO Reason=Explicacion
```

scontrol > show node

El comando aporta bastante información: recursos reservados por Slurm, OS, kernel, uptime, uso actual de memoria...

Métricas memoria:

RealMemory memoria total, según se declara en slurm.conf
AllocMem memoria reservada por trabajos
FreeMem memoria libre (según el sistema). Corresponde con la métrica "free" de top (que no toma en cuenta el uso para cache)

scontrol > ejemplo show node

```
NodeName=gpu02 Arch=x86_64 CoresPerSocket=16
   CPUAlloc=25 CPUEfctv=32 CPUTot=32 CPULoad=10.02
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=gpu:hopper:1
   NodeAddr=gpu02 NodeHostName=gpu02 Version=22.05.8
   OS=Linux 5.14.0-362.13.1.el9_3.x86_64 #1 SMP PREEMPT_DYNAMIC Fri Nov 24 01:57:57 EST 2023
   RealMemory=514000 AllocMem=409600 FreeMem=292525 Sockets=2 Boards=1
   State=MIXED ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=gpu
   BootTime=2024-11-11T16:13:39 SlurmdStartTime=2025-01-03T10:26:44
   LastBusyTime=2025-02-05T10:27:41
   CfgTRES=cpu=32,mem=514000M,billing=32,gres/gpu=1
   AllocTRES=cpu=25,mem=400G,gres/gpu=1
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Slurm > contabilidad

Contabilidad > Introducción

OS:

- Usuario
- Grupo

Slurm:

- Usuario
- Cuenta (`Account`)
- Asociación = cluster + account + usuario

Contabilidad > sacct

Comando para obtener información sobre trabajos ejecutados

sacct > parámetros

-o qué información mostrar
-j limitar consulta a trabajos específicos
-u limitar consulta a un usuario
-S buscar a partir de una fecha

sacct > información (campos) básicos

- jobname
- start, end, elapsed
- memoria: maxrss, averss

sacct > campos

```
jobid
jobname
state
start, end, elapsed
SubmitLine
Comment
cputime
user
group
```

```
nmtasks
alloccpus
+allocgres+
allocnodes
nodelist
reqcpus, reqmem, reqnodes
maxrss, averss
maxdiskread, maxdiskwrite
```

sacct > ejemplos (I)

```
sacct
sacct --helpformat

sacct -o jobid,jobname,elapsed,alloccpus,maxrss

sacct -o jobid,jobname,user,state,start,end,alloccpus,\
  alloctres,allocnodes,nodelist,cputime

sacct -j JOB_ID1,JOB_ID2
```

sacct > ejemplos (II)

```
export SACCT_FORMAT="jobid,jobname,reqmem,averss,maxrss,avecpu,cputime"
sacct
```

```
# -o / --format: %ANCHO (justif. dcha, izda     %-ANCHO)
sacct -o jobid,nodelist%30

sacct --accounts=deeplearning

sacct -S$(date -d "14 days ago" --iso) -u afont
```

Contabilidad > sacctmgr

Control

associations limits limits,qos

sacctmgr / "Grupos"

```
sacctmgr add account research cluster="hpc"  Description="All researchers" \
   Organization="Supercomputing"

sacctmgr add account chem Description="Chemistry" parent="research"

sacctmgr show account

sacctmgr show account -s
sacctmgr show account format=account,descr%40,org
sacctmgr show account -s format=user,account,qos,tres,grptres%30

sacctmgr modify account where name=research set Description="Desc"
```

sacctmgr / Usuarios

DefaultAccount: account por defecto (se puede pertenecer a varias)

```
sacctmgr add user afont DefaultAccount=research
sacctmgr modify user afont DefaultAccount=ml

sacctmgr show user
sacctmgr show user format=user%40,admin%14
sacctmgr show user -s USER
```

sacctmgr / Usuarios-grupos

```
# afont-research (default), afont-deeplearning
sacctmgr add user afont Account=deeplearning
# Volver a solo "research"
sacctmgr delete user afont Account=deeplearning
```

(lo que se crean / borran son asociaciones usuario-account)

slurm > Prioridades & QOS

Conceptos > Prioridad

Valor entero asociado a un trabajo. Cuanto mayor prioridad tenga un trabajo, antes será elegido en la cola para entrar en ejecución (una vez haya suficientes recursos)

Prioridades / Orden básico

Los trabajos se escogen siguiendo estos criterios (por orden)

1. Reservas
2. Prioridad `PriorityTier` (de una cola respecto a otra)
3. Prioridad `PriorityJobFactor` (multifactor)
4. Cuando se envió el trabajo

Es decir, no siempre gana quien envía antes, o la mayor prioridad

Prioridades / colas

```
# Max: 65533
PartitionName=hpc  Nodes=... PriorityTier=10
PartitionName=fast Nodes=... PriorityTier=20
```

Conceptos > QoS

Una vez definida (admin) se solicita para cada trabajo

Permite controlar...

- prioridad de ejecución (`PriorityWeightQOS`, vía multifactor priority)
- cuotas (límites) recursos

Al crear un cluster se añade una QOS por defecto: `"normal"`

Prioridades / Multifactor priority

Factores de la fórmula:

- `Age`: tiempo de espera
- `Job size`: nodos o CPUs solicitadas
- `TRES`
- `QoS`
- `Partition`
- `Fair-share`: reparto equitativo (por cuentas / grupos)

Multifactor priority/Fórmula

```
Job_priority =
   (PriorityWeightAge) * (job_age_factor) +
   (PriorityWeightFairshare) * (job_fair-share_factor) +
   (PriorityWeightJobSize) * (job_size_factor) +
   (PriorityWeightPartition) * (PriorityJobFactor) +
       (PriorityWeightQOS) * (QOS_factor) +
   SUM(TRES_weight_cpu * job_TRES_factor_cpu,
       TRES_weight_[type] * job_TRES_factor_[type>],
       ...)
```

```
# PriorityWeightTRES=CPU=1000,Mem=2000,GRES/gpu=3000
```

Multifactor / ejemplo

```
PriorityType=priority/multifactor

PriorityWeightFairshare=10000
PriorityWeightAge=1000
PriorityWeightPartition=0
PriorityWeightJobSize=0

# Cuanto tiempo atrás considerar (por defecto, 7 días)
PriorityDecayHalfLife=10-0
# Cuándo se maximiza el factor age (por defecto, 7 días)
PriorityMaxAge=3-0
```

Slurm > Límites

Limitar uso recursos

1. QOS cola
2. QOS trabajo
3. Usuario
4. Grupo (account)
5. Grupo "root"
6. Cola

Límites > AccountingStorageEnforce

```
# associations / limits / qos
AccountingStorageEnforce=limits
```

QOS y limits implican associations (que a su vez implica que los usuarios necesitan estar dados de alta en Slurm...)

Límites > parámetros

Contexto se aplica a...

- "Cola": Partition
- Cuenta: Account
- "Calidad de servicio": QOS

Momento se aplica en el envío (`Submit`) o en la cola

Ámbito una cuenta individual, o una "agrupación" (`Grp`, que incluye a sus subgrupos)

Límites > Resumen

```
(A)    MaxJobs (per user)          (A)(Q) GrpTRES (GrpCPUs, GrpMem)
(A)    MaxSubmitJobs               (A)(Q) GrpTRESMins
(Q)    MaxJobsPerAccount           (A)(Q) GrpJobs
(Q)    MaxSubmitJobsPerAccount     (P) MaxMemPerCPU
(Q)    MaxTRESPerAccount           (P) MaxMemPerNode
(Q)    MaxTRESPerUser              (P) MaxNodes
```

```
(Q)     MaxJobsPerUser              (P) MaxCPUsPerNode
(Q)     MaxSubmitJobsPerUser        (P) MaxTime
(A)     MaxTRES                     (A)(Q) MaxTRESMinsPerJob
(A)(Q) MaxTRESPerJob                (A)(Q) MaxWallDurationPerJob
(Q)     MinTRESPerJob
```

A Account, Q QOS, P Partition, Mins minutos, TRES Trackable RESources

Límites > tiempo

En script:

```
# duración esperada: 1 hora
#SBATCH --time=01:00:00
```

En slurm.conf...

```
# A nivel de partición (10 días)
PartitionName=hpc Nodes=... MaxTime=10-0 ...

# Cuantos minutos puede sobrepasar un trabajo el límite de tiempo
# antes de ser cancelado
OverTimeLimit=60
```

Formatos especificación tiempo

- minutes
- minutes:seconds
- hours:minutes:seconds
- days-hours
- days-hours:minutes
- days-hours:minutes:seconds
- ”UNLIMITED”

Parámetros > Particiones > CPU y memoria

Por defecto:

- DefMemPerCPU (MB)
- DefMemPerNode

Máximo (límite)

- MaxMemPerCPU
- MaxMemPerNode

(por trabajo)

Límites > usuarios / grupos

Límites asociados a usuario

- MaxJobs
- MaxTRESPerJob
- MaxWallDurationPerJob

```
sacctmgr modify user afont set maxjobs=2

sacctmgr modify user afont set MaxTRES=Node=4

# numero CPUs / MB
sacctmgr modify account research set GrpTRES=cpu=1,mem=8000

sacctmgr show account -s format=user,account,qos,tres,grptres

sacctmgr show association
```

Límites > QoS

QoS > asociar a usuarios

QOS(=,+,-): asignar / añadir / quitar QoS

```
sacctmgr add qos small
sacctmgr show qos
sacctmgr modify user afont set qos=small
sacctmgr modify user afont set qos+=normal
sacctmgr delete qos small
```

```
sbatch --qos=QOSNAME ...
```

QoS > definir límites

```
sacctmgr modify qos small set priority=20
sacctmgr modify qos small set MaxCPUs=1
sacctmgr modify qos fast set MaxTRES=cpu=32
# para quitar un límite, usar el valor -1
sacctmgr modify qos small set MaxCPUs=-1
sacctmgr modify qos fast set MaxTRES=cpu=-1
```

—

Buenas prácticas

Guía del buen usuario

Recomendaciones (I)

Buenas prácticas / recomendaciones generales

- comienza por lo simple: entrar al nodo de login.
- el cluster es versatil: si necesitas hacer un uso interactivo y/o gráfico de los nodos de cálculo, puedes hacerlo (como viste en la demostración). En el caso gráfico, el uso directo de ventanas suele ser más que suficiente.
- si tu ordenador es windows, prueba Moba Xterm (suele ser la opción más fácil/amigable)

Recomendaciones (II)

- en general, todo lo que se ejecuta como usuario "no privilegiado" solo afecta al usuario en particular. Prueba y experimenta (siendo consciente del uso de recursos)
- conoce tu aplicación científica, para no desperdiciar recursos (memoria, GPUs...) que podría utilizar otra persona

Recomendaciones (III)

Cuidado con los parámetros (por defecto)

- se explícit@. Cuanta más información demos (por ejemplo, parámetros de Slurm), mejor. Los parámetros por defecto a veces no son los que creemos.
- Caso slurm:
  - indicar (estimación) de los parámetros –time, –mem (o –mem-per-cpu)
  - indicar –ntasks (o -n) y –gres (cuando se necesite GPU): -n 1 o –gres=gpu:1
  - delegar en Slurm la búsqueda de los recursos en los nodos

Recomendaciones (VI)

Almacenamiento:

- por defecto, lo simple. Usar el espacio dentro del directorio de usuario (en BeeGFS, no en /home)
- si hay tiempo, probar a usar el scratch y comparar tiempos de ejecución
- si vas a subir/bajar muchos datos al cluster, prueba con rsync

Recomendaciones (VII)

Paralelización:

- de nuevo, comenzar por lo simple. Lanzar varios trabajos (cada uno ejecutando un único comando de una app que no es paralela) ya supone una gran ganancia en tiempo de ejecución
- si hay tiempo, ir probando distintos parámetros (en base a las posibilidades de la aplicación) y comparar. Ejemplo: si es una aplicación multihilo, probar a usar 4, 8, 16 hilos... y ver como evoluciona el tiempo de ejecución
- profundizar en los aspectos técnicos de la aplicación científica que se quiere usar. Ejemplo: quizá soporta AVX512 y se puede optimizar su binario para que vaya más rápido.

GESTIÓN DE RECURSOS

¿Hay alguna forma a priori de saber cuánta cpu y/o gpu necesita el archivo que voy a cargar?

A priori un comando no lo puede saber. Se puede hacer una estimación cuando se conoce el algoritmo que usa el programa.

Una aproximación es hacer una primera ejecución solicitando muchos recursos, y luego usar el comando `sacct` para determinar los que realmente se han empleado.

(también se puede usar herramientas como top, pero solo dan una visión momentánea del consumo)

Uso interactivo

Tradicionalmente pensamos los "sistemas de colas" como no interactivos (batch)

Pero Slurm también permite uso interactivo, tanto en consola como con ventanas gráficas (con la ventaja de gestionar los recursos que cada cual necesite, automáticamente)

GESTIÓN DE DATOS

Copias de seguridad

Ideal:

- sistema de copia automático (requiere almacenamiento independiente, sea cinta o disco)
- soluciones basadas en software abierto se pueden conseguir a un coste razonable

Alternativas:

- delegar en cada usuario.
- mejor algo simple (como `rsync` [1]) que no tener backups

[1] aunque no sea una herramienta de backup propiamente dicha

LOGS

Logs aplicación

(dependen de cada aplicación...)

La salida estándar de trabajos en Slurm se puede ver en los ficheros `.out` / `.err`

Logs sistema

Eventos sistema operativo / servicios

Dentro de `/var/log`...

- Slurm: `slurm/slurmd.log`
- RedHat / Rocky / ...:
  - `messages`
  - `secure`
- Debian / Ubuntu / ...
  - `syslog`
  - `auth`