



Universidad de Alcalá

UNIVERSIDAD DE ALCALÁ

Slurm guide

USER AND ADVANCE GUIDE FOR CLUSTER SLURM MANAGER

Raúl De Armas Rodríguez

Departamento de Química Analítica, Química Física e Ingeniería Química

21 de octubre de 2022

1. Introduction.

En Química computacional es necesario el uso de cluster para realizar cálculos en paralelo dado que muchos de estos cálculo son extremadamente costosos a nivel de tiempo y energía gastada. Por ello es usual correr estos cálculos en paralelo, es decir, dividiendo las tareas de un programa en varios fragmentos que se realizaran por separado, para luego juntarse y obtener el resultado final. En un computador de mesa podemos encontrar de forma normal que tenemos 1 nodo, es decir una unidad de procesamiento (por ejemplo un procesador Intel i7-12700) que contiene de forma usual un cierto núcleos(core) que son básicamente subprocesadores independiente (para el i7 mencionado tenemos 12 cores. Cosa aparte, tenemos los hilos(threads) que es algo distinto; siendo un hilo el número máximo de procesos que se pueden realizar, pero no son independientes como con los cores, si no que cada proceso se divide en fragmentos y se van ejecutando todos los fragmentos en los distintos núcleos(core) de forma organizada para obtener un mejor rendimiento(performance).

En un cluster de computación lo que tenemos es un grupo de ordenadores conectados entre ellos (imagínese un montón de ordenadores apilados, pero quítele toda la carcasa y quédese solo con la *Central Processing Unit*, CPU) para realizar un calculo entre todos ellos. Por tanto 3 ordenadores serían tres nodos en la terminología que usaremos, si cada nodo tiene 8 subprocesadores, cores a partir de ahora, tendremos 24 cores para paralelizar el cálculo.

¡Pero claro! ¿Como gestionamos la comunicación entre todos estos procesadores? Pues para ello hay múltiples herramientas que controlan la comunicación entre los procesadores para que se coordinen entre ellos y se puedan comunicar; uno de los más famosos y utilizados es [MPI](#) de *Message Passing Interface*. Pero, lo que realmente nos interesa es que no siempre queremos utilizar todos los nodos y todos los cores que tenemos en nuestro cluster, supongamos que tenemos 6 ordenadores nombrados tal que n1, n2, n3 ... En nuestro cluster queremos tener una parte del cluster, del n1-n4 para calcular ecuaciones diferenciales y otro parte, del n5-n6, para calcular numero primos, y los queremos tener completamente separados. Así pues lo que usaremos para crear estas dos subsecciones es un sistema de colas.

Un sistema de colas (Queue Manager) se encarga de gestionar los trabajos que mandamos a los nodos (la comunicación entre estos recuerda que la realiza MPI u otro). Sería el encargado de dividir entre nuestro 6 nodos en dos particiones a las que llamaremos colas. Te preguntaras que por que las llamamos colas, la respuesta es por que cada partición (cola) actuara como un lista de trabajos. Es decir supongamos que mando a la cola1 (que contiene los nodos n1-4) un calculo y me voy a casa; este puede tardar 5 minutos, 6 horas o 3 semanas. Entonces yo quiero que sea cuando sea que termine empiece con otro calculo que tengo preparado; esto es lo que hace el sistema de colas. Es decir, voy mandando trabajos y se van poniendo uno tras otro en espera, y cuando uno termina el siguiente empieza.

¿Nos falta algo? Pues sí, si por ejemplo en la cola1 (recuerda que son 4 de los 6 ordenadores/nodos conectados entre ellos) tenemos por cada nodo 8 cores, tenemos en total 32 cores. Por lo que podemos manda un calculo de 32 cores, 2 cálculos de 16 o 4 cálculos de 8 cores; también existe la posibilidad de mandar uno de 22 (el número de procesadores puede ser cualquiera en realidad) y otro de 8 procesadores, quedándonos 2 cores huérfanos. Pues el sistema de colas se encarga de que ordenar también estos cálculos con una cierta prioridad, es decir si quedan dos nodos sueltos y yo me pongo a la cola con 8 cores, pues no hay hueco para mi, pero si después pido hueco para un calculo de dos cores, me saltare la cola (¡sin pedir perdón ni nada!) y mi cálculo de 2 cores entrara antes que el de 8. Este tipo de administración y otras mas avanzadas es de los que se encarga un sistema de colas.

1.1. Slurm

[Slurm](#) de *Simple Linux Utility for Resource Management* es un gestor de clusters y planificador de trabajo para Linux de fuente abierta (Open Source). Sus tres funciones principales,

1. Asignar el acceso a los recursos (nodos de calculos) de forma exclusiva y/o no exclusiva a los distintos usuarios del cluster.
2. Proporcionar un entorno de trabajo para correr, ejecutar y monitorear los trabajos enviados.

3. Gestionar/Arbitrar las demandas de los recursos organizando una cola de trabajos.

1.2. Slurm Architecture

Definición: Daemon

Daemon o Demonio (de *Disk And Execution Monitor*) es un tipo de programa que se ejecuta en segundo plano y sin control directo del usuario; pudiendo ejecutarse desde el arranque, de forma persistente o auto-reiniciarse. Un ejemplo de ello es cuando recibes un correo, de forma persistente hay un daemon que pregunta al servidor de correos si hay algún correo nuevo, no esta el usuario refrescando continuamente el correo para ver si ha llegado algo.

En slurm lo que tenemos es un Daemon llamado **slurmd** que corre en cada nodo y otro daemon, llamado **slurmctld**, corriendo en modo administrador. Es decir que tenemos un programa principal, **slurmctld**, que gestiona todos los nodos, y dentro de cada uno de estos hay otro programa, **slurmd** que gestiona cada nodo.

También nos permite (y es lo usual) que las colas se superpongan. Cogiendo el ejemplo de los 6 ordenadores (nodos) conectados, esto implicaría que podemos tener 4 nodos en una cola [n1-4] además de las mitad de los cores de el n5; teniendo 8 cores por nodos esto implicaría tener una partición (cola) de 36 cores y otra de 12 (n6+1/2n5). Otras funciones que se puede añadir es limitar el tamaño y número de trabajos, el máximo tiempo que puede correr un cálculo, la cantidad máxima de cores o memoria por trabajo y mucho más.

1.3. ¿Que necesitamos para lanzar un cálculo?

Tan solo necesitamos un archivo en formato bash, es decir terminado en *.sh*. Dentro de este archivo habrá unas instrucciones para el gestor de colas (es decir el programa slurm) y aparte las ordenes a ejecutar. En esta guía no se entrara en detalle de como se componen estos archivos pues hay multitud de ejemplos en internet, lo que si añadiremos serán unos ejemplos más adelante.

También puedes encontrar ejemplos e información en:

<https://github.com/statgen/SLURM-examples/blob/master/README.md> https://help.rc.ufl.edu/doc/Sample_SLURM_Scripts

2. Comandos para el usuario

2.1. Comandos básicos

Algunos comandos básicos para el usuario inexperto.

2.1.1. --help

En general podemos usar detrás de cualquier comando esta opción para que nos devuelva en la terminal información de un comando en específico. Por ejemplo, podemos querer saber que opciones tenemos dentro del comando **sbatch**, entonces escribiríamos en la terminal.

```
sbatch --help
```

Esto nos devolvería toda la información impresa en la terminal. Si quisiéramos ver la información pero no imprimirla en la terminal (más cómodo para cuando el output del `--help` es muy largo) podemos usar el siguiente comando para leer el texto en un formato parecido al que tiene el editor de texto *vim*.

```
sbatch --help | less
```

2.1.2. sbatch

Este comando es con el que lanzaremos el cálculo, es decir una vez tengamos nuestro script para ejecutar el sistema de colas (ir a 1.3) haremos,

```
sbatch sub.sh
```

Donde `sub.sh` es el script de `slurm`. A esta opción podemos añadirle banderas que deseamos para cambiar el script *in-situ* o si estamos haciendo auto-scripto (script que generan scripts) puede ser nos muy útil. Las principales flags a añadir son:

1. `--job-name={nombre del trabajo}`
2. `--mem={memoria que demandamos para el trabajo}` (ex:4G)
3. `--time={tiempo límite del trabajo}` en formato `{days}:{hours}:{minutes}` (ex:--time=3-2:30)
4. `--partition={partition name}` podemos ver los nombre con el comando *sinfo* (ir a 2.1.6)
5. `--output={nombre del archivo del stdout y stderr}`

2.1.3. `squeue`

Indica el estado de los trabajos enviados y tiene multitud de opciones de filtrado, orden y formato. Por defecto esta ordenado por orden de prioridad. Algunas opciones importantes son:

1. `-a, --all=`
Imprime los trabajos de todos los usuarios y en todas las particiones, incluso las que no son accesibles.
2. `-u, --user=`
Imprime los trabajos por usuario, si necesita ver el de todos los usuarios usar comandos comodín (wildcard) como el `"*"`.
3. `-v, --verbose`
Imprime los trabajos con más información.
4. `-O, --format=`
Especifica la información a ser impresa (la `"-o"` minúscula es lo mismo pero usa letras tal que `%Z` en vez de palabras enteras). Hay multitud de opciones que vienen descritas en la documentación de `squeue`.

2.1.4. `scancel`

Permite matar los trabajos que esten corriendo. Se mata en base al *JOBID* (el cual podemos ver usando *squeue*); suponiendo que el *JOBID* es 321,

```
scancel 321
```

Otras opciones mas avanzadas pueden verse en 4.2.

2.1.5. `sacct`

Este comando es de mucha utilidad pues contiene un registro de información de los trabajos que están corriendo y los anteriormente enviados; por lo que es muy util para tener un registro de lo realizado. Algunos banderas útiles son,

1. `-f, --file=`
Selecciona la información por nombre del trabajo.
2. `-E, --endtime`
Selecciona los trabajos que terminasen antes de la hora propuesta, un ejemplo de esto sería,

```
sacct -E 13:00
sacct -E 10/20-13:00
```

Respectivamente para los trabajos que terminasen antes de las 13:00 de hoy y para los trabajos terminados antes del día 20 de octubre (mes 10) a las 13:00 horas. En la documentación podremos encontrar otras opciones muy útiles, entres las que se incluyen *today* o *midnight*.

Podemos encontrar información mas avanzada en 4.3.

2.1.6. **sinfo**

Nos informa del estado de las particiones y de los nodos. Tiene muchas funciones de filtrado, ordenado y formateo. En sí la información más básica y útil aparece ejecutando el comando sin ninguna bandera, la información tiene la estructura que aparece en la figura 4, podemos encontrar más información al respecto de cada propiedad en 4.1.

Algunas banderas útiles son:

1. **--all**
Imprime aun más información de cada partición (cola)
2. **-N**
Nos formatea la información a una lista por nodo, es decir si por ejemplo utilizamos la bandera **-R** podemos añadir la bandera **-R** para que nos especifique la información por nodo y nos por partición. Esto lo vemos en la figuras 1 y 2.
3. **-R**
Nos informa de las razones por las que los nodos no funcionan.
4. **-s**
Resume la información.

```
REASON      USER      TIMESTAMP      NODELIST
NO NETWORK ADDRESS F root      2022-03-26T21:19:56 nodo[02-04]
```

Fig. 1: Ejemplo de salida del comando *sinfo -R*

```
NODELIST    NODES    PARTITION    STATE
nodo02      1        ladon*       down*
nodo03      1        ladon*       down*
nodo04      1        ladon*       down*
```

Fig. 2: Ejemplo de salida del comando *sinfo -R -N*

2.1.7. **scontrol**

El comando **scontrol** se utiliza para ver y modificar la configuración de slurm. La mayoría de los comandos solo puede ser utilizados por el administrador (root). Algunos comandos útiles,

1. **scontrol show jobid *JOBID***
Imprime información detallada del un trabajo por su ID.

2.1.8. **all.q**

Si usted es usuario de nuestro cluster puede usar este comando para obtener información de los recursos disponibles y utilizados por nodo.

Si no es usuario de nuestro cluster puede encontrar este código para slurm 21.08.1 en el [repositorio de github](#) autor de este texto.

2.1.9. Para migrantes de SGE:

Puede encontrar más información en el recurso:

1. <https://hpcsupport.utsa.edu/foswiki/pub/Main/SampleSlurmSubmitScripts/SGEtoSLURMconversion.pdf>.
2. También (si es usted usuario de nuestro cluster podrá encontrar junto a este documento en *./SGE-toSLURM/SGEtoSLURMconversion.pdf*).

| NodeName | CPU[use/tot/%] | Mem[use/tot/%] | MemLimit |
|----------|----------------|----------------|----------|
| nodo01 | 0/64/0,00% | 0/185/0,00% Gb | 4 Gb |
| nodo02 | 0/32/0,00% | 0/250/0,00% Gb | 4 Gb |
| nodo03 | 0/32/0,00% | 0/250/0,00% Gb | 4 Gb |
| nodo04 | 0/32/0,00% | 0/250/0,00% Gb | 4 Gb |
| nodo05 | 0/40/0,00% | 0/92/0,00% Gb | 4 Gb |
| nodo06 | 0/32/0,00% | 0/250/0,00% Gb | 4 Gb |

Fig. 3: Ejemplo de salida del comando `all.q`

| Utilidad | SGE | Slurm |
|-------------------|----------|----------|
| Job submission | qsub | sbatch |
| Job status by job | qstat | squeue |
| Job deletion | qdel | scancel |
| List nodes | qhost | sinfo -N |
| Cluster status | qhost -q | sinfo |
| GUI | qmon | sview |

Cuadro 1: Conversion de comandos basicos entre SGE y SLURM

3. Ejemplos de archivos de input para slurm

3.1. Ejemplos generales

Podemos encontrar multiples ejemplos en,
<https://hpc-uit.readthedocs.io/en/latest/jobs/examples.html>
https://support.cecil-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html

De estas direcciones anteriores se han sacado los ejemplos detallados a continuación.

3.1.1. Message passing example (MPI)

```
#!/bin/bash
#
#SBATCH --job-name=test_mpi
#SBATCH --output=res_mpi.txt
#
#SBATCH --ntasks=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

module load OpenMPI
srun hello.mpi
```

3.1.2. Shared memory example (OpenMP)

```
#!/bin/bash
#
#SBATCH --job-name=test_omp
#SBATCH --output=res_omp.txt
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK
srun ./hello.omp
```

3.1.3. Con multi-procesamiento(MPI) y multithreading (OpenMP)

```
#!/bin/bash
#
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=4
module load OpenMPI
export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK
srun ./myprog
```

3.1.4. GPU jobs

```
#!/bin/bash
#SBATCH --job-name=example
#SBATCH --ntasks=1
#SBATCH --time=1:00:00
#SBATCH --mem-per-cpu=1000
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1

module load CUDA

srun ./my_cuda_program
```

3.2. Ejemplos gaussian 16

Este script de ejecución fue escrito por Alain Arsene.

```
#!/bin/bash -l
#####
#####
#                               #
#   The SLURM directives   #
#                               #
#####
#
#           Set number of resources
#
#SBATCH --comment=gaussian_job.
#SBATCH --job-name=gaussian_job.
#SBATCH --output=output.%j.gaussian_job.
#SBATCH --error=output.%j.gaussian_job.
##SBATCH --gres=gpu:1 # Number of gpu
#SBATCH -n 8          # Number of CPU
#SBATCH --mem= 8GB
#SBATCH -p ladon
#SBATCH -N 1

module purge
module load gaussian/16-avx2

export HOMEDIR=$SLURM_SUBMIT_DIR
export SCRATCHDIR=/scratch/job.$SLURM_JOB_ID.$USER
export WORKDIR=$SCRATCHDIR
mkdir -p $WORKDIR

cd $HOMEDIR
cp gaussian_job.com $WORKDIR
cp checkfilename.chk $WORKDIR 2> /dev/null

cd $WORKDIR
g16 < gaussian_job.com > gaussian_job..log
wait
cp $WORKDIR/*.7 $HOMEDIR
cp $WORKDIR/${chkFileName}.chk $HOMEDIR
cp $WORKDIR/gaussian_job..log $HOMEDIR
wait
rm -rf $WORKDIR
```

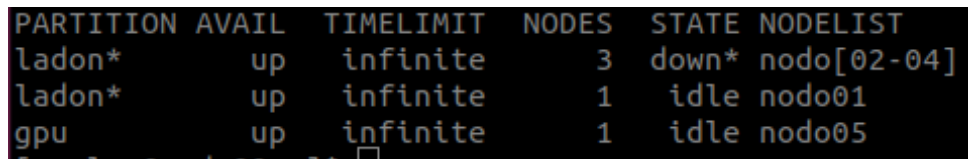
4. Comandos e información avanzada.

4.1. Mas información de [sinfo](#)

Si realizamos el comando

```
sinfo
```

Nos aparecerá algo parecido a lo que vemos en la figura 4. Como vemos tenemos el nombre de la partición, uso, tiempo límite, nodos, estado, lista de nodos.



| PARTITION | AVAIL | TIMELIMIT | NODES | STATE | NODELIST |
|-----------|-------|-----------|-------|-------|-------------|
| ladon* | up | infinite | 3 | down* | nodo[02-04] |
| ladon* | up | infinite | 1 | idle | nodo01 |
| gpu | up | infinite | 1 | idle | nodo05 |

Fig. 4: Ejemplo de salida del comando `sinfo`.

4.1.1. Partition

Respecto al **Partición** | **Partition**: son los nombres de las colas que están disponibles para enviar. En la figure 4 vemos que tenemos una llamada `gpu` y dos que se llaman `ladon`; el asterisco que aparece junto al nombre indica (*la siguiente información no fue encontrada en el manual y por tanto puede no ser totalmente verídica*) que es la partición por defecto (default) para los trabajos enviados.

4.1.2. Avail

Respecto al **Uso** | **Avail** podemos tener los estados:

1. **up**: todo bien.
2. **down**: El nodo para todos los trabajos y ningún trabajo de la cola se ejecutara.
3. **drain**: El trabajo enviado esta corriendo pero ningún otro más se ejecutará. Esto puede ser por que entre en mantenimiento o por algun error al ejecutar el trabajo.
4. **inact**: El nodo esta inactivo, no es accesible.

4.1.3. State

Respecto al **Estado** | **State** podemos tener múltiples estados, podemos encontrar más información en <https://slurm.schedmd.com/sinfo.html>. Los estado más importante son:

1. **idle & alloc**: Son estado de un nodo "sano", respectivamente implican sano y sin ejecutar trabajos; y sano ejecutando trabajos.
2. **down & idle***: Son estados de un nodo enfermo, cuando al aparece en un estado ***** es que el nodo no es accesible y no recibirá ningún trabajo, si permanece en este estado pasará a estado **down**

4.2. Mas información de `scancel`

En la documentación de `scancel` podemos encontrar todas las opciones. En esta sección solo comentaremos las más interesantes.

1. **--name=**
Mata el trabajo por su nombre.
2. **-t, --state=**
Restringe la operación por estado del trabajo. Siendo los posibles estados

PENDING, RUNNING o SUSPENDED

Por ejemplo si quisiéramos matar todos los trabajos pendiente de nuestro usuario podemos hacer tal que,

```
scancel -t PENDING -u <username>
```

3. **--me**
Restringe la operación a mi usuario actual.
4. **-A, --acount=**
Restringe la operación al usuario seleccionado (si no eres root no podrás matar los otros trabajos).

4.3. Mas información de [sacct](#)

```
sstat -format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j %jobid%--allsteps
```

4.4. Otros comandos de utilidad

Para obtener información de un trabajo que esta corriendo.

```
sstat --format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j <jobid> --allsteps
```

Para trabajos que ya han terminado podemos obtener información del trabajo por ID tal que,

```
sacct -j <jobid> --format=JobID,JobName,MaxRSS,Elapsed
```

Para obtener información de todos los trabajos enviados por usuario.

```
sacct -u <username> --format=JobID,JobName,MaxRSS,Elapsed
```