Límites de memoria Slurm

Slurm impone un límite de memoria en cada trabajo. De forma predeterminada, es deliberadamente relativamente pequeño: 2 GB por nodo. Si su trabajo usa más que eso, recibirá un error que indica que su trabajo ha excedido la memoria solicitada (*Exceeded job memory limit*).

Para establecer un límite mayor, agregue a su envío de trabajo:

```
#SBATCH --mem X
```

donde X es la cantidad máxima de memoria que usará su trabajo por nodo, en MB.

Cuanto mayor sea su conjunto de datos de trabajo, mayor deberá ser, pero cuanto menor sea el número, más fácil será para slurm encontrar un lugar para ejecutar su trabajo.

Para determinar un valor apropiado, consulte siguiente sección.

Cómo estudiar la eficiencia de un job

Muchas veces nos preguntamos cómo puedo saber qué tan eficiente es mi trabajo y para ello tenemos el comando ***seff***.

```
seff JOBID
```

donde el JOBID es el id del trabajo en el que estás interesado. Por ejemplo:

```
$ seff 1553

Job ID: 1553
Cluster: teide
User/Group: viddata/viddata
State: COMPLETED (exit code 0)
Nodes: 4
Cores per node: 16
CPU Utilized: 2-11:55:56
CPU Efficiency: 89.73% of 2-18:47:28 core-walltime
Job Wall-clock time: 01:02:37
Memory Utilized: 26.08 GB (estimated maximum)
Memory Efficiency: 23.85% of 109.38 GB (27.34 GB/node)
```

Solicite memoria un poco mayor a lo que *seff* dice.

Debe configurar la memoria que solicita en algo un poco más grande que lo que dice seff.

***seff* informa del máximo de memoria usada en jobs paralelos.**

Tenga en cuenta que **para trabajos paralelos que abarcan varios nodos, esta es la memoria máxima utilizada en cualquier nodo**; si no está configurando una distribución uniforme de tareas por nodo (por ejemplo, con --ntasks-per-node), el mismo trabajo podría tener valores muy diferentes cuando se ejecuta en diferentes momentos.

Espera a que el trabajo termine satisfactoriamente.

También tenga en cuenta que el número registrado por slurm para el uso de la memoria será inexacto si el trabajo finalizó de forma no satisfactoria. **Para obtener una medición precisa, debe tener un trabajo que se complete correctamente, ya que slurm registrará el pico de memoria real**.

Memoria máxima por tipo de nodo

Tipo de nodo	Solicitud máxima de memoria en slurm	
Sandy bridge 16 Cores - 32 GB	30000 MB	
Sandy bridge 16 Cores - 64 GB	62000 MB	
Fat Nodes 32 Cores - 256 GB	254000 MB	bajo solicitud
Icelake Nodos GPU 64 Cores - 256 GB	254000 MB	bajo solicitud

¿Cómo puedo saber qué tan eficiente es mi trabajo?

Puede ver la eficiencia de su trabajo comparando MaxRSS, MaxVMSize, Elapsed, CPUTime, NCPUS con el comando *sacct*.

```
sacct -j 999997 --
format=User,JobID,Jobname%25,partition,elapsed,MaxRss,MaxVMSize,nnodes,ncpus,nodelist%20
```

```
   User       JobID         JobName  Partition    Elapsed    MaxRSS  MaxVMSize   NNodes
NCPUS          NodeList
--------- ------------ ------------------ ---------- ---------- ---------- ---------- -------- ---------- --------------------
 eolicase 999996        WRF_2023080618     batch  13:32:47                    4       64     node1511-
[1-4]
       999996.batch          batch          13:32:47   216700K   815628K     1       16       node1511-1
       999996.0         geogrid.exe          00:00:57   236240K  2762564K     4        8     node1511-
[1-4]
       999996.1         metgrid.exe          00:01:54    15395M   591912K     4        4     node1511-
[1-4]
       999996.2           real.exe           00:00:26   338148K  2928876K     4       16     node1511-
[1-4]
       999996.3            wrf.exe           13:26:36   598864K  3768748K     4       64     node1511-
[1-4]
```

En este trabajo, ve que el usuario usó 64 núcleos y su trabajo se ejecutó durante 13,5 horas. Sin embargo, su tiempo de CPU es de 13,5 horas, que está cerca de las 64*13 horas. Si su código escala efectivamente según esta fórmula *CPUTime = NCPUS * Elapsed* su aplicación escala perfectamente. Si no lo es, el resultado divergirá de la fórmula y la mejor manera de probar esto y determinar cómo escala la aplicación es hacer algunas pruebas de escala.

Hay dos formas de hacer esto: ***Strong scaling*** (escalado fuerte) and ***Weak scaling*** (escalado débil).

***Strong scaling* (Escalado fuerte)**

El escalado fuerte es donde deja el tamaño del problema igual pero aumenta la cantidad de núcleos. Si su código se escala bien, debería tomar menos tiempo proporcional a la cantidad de núcleos que usa.

***Weak scaling* (escalado débil)**

La cantidad de trabajo por núcleo sigue siendo la misma, pero aumenta la cantidad de núcleos, por lo que el tamaño del trabajo se escala proporcionalmente a la cantidad de núcleos. Por lo tanto, si su código se escala en este caso, el tiempo de ejecución debería seguir siendo el mismo.

Por lo general, la mayoría de los códigos tienen un punto en el que la escala se rompe debido a ineficiencias en el código.

Por lo tanto, más allá de ese punto, no hay ningún beneficio en aumentar la cantidad de núcleos que arroja al problema. Ese es el punto que quieres buscar. Esto se ve más fácilmente trazando el registro de la cantidad de núcleos frente al registro del tiempo de ejecución.