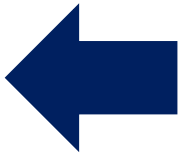
Sugerencias para el envío de trabajos en el HPC UO

Oscar Au Alvarez , 24 Enero 2020

RECURSOS DEL HPC UO

HPC UO

- **Clúster homogéneo: 13 nodos de cálculo (nodo001, nodo002,…, nodo013)**
- **Cada nodo: 32 cores, Memoria 64 Gb**



- **Conexión: Infiniband 40 Gbps**
- **No GPU (por ahora)**

MÓDULOS

Trabajo con módulos

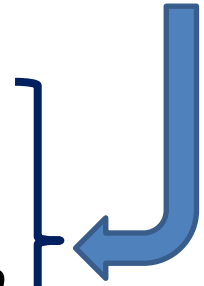
- **Todo el trabajo con aplicaciones se hace con módulos**
- **Hay que incluirlos en el script de envío**

- **Permiten el cambio *dinámico* de las variables de ambiente**
- **Declaran: camino de ejecutables , bibliotecas, otros módulos que se cargan, etc**

Trabajo con módulos

- *module available (ml av)*
- …….
- GCC/7.3.0-2.30
- GCC/8.2.0-2.31.2
- GCC/8.3.0

En script



- *module load GCC* # carga la última versión
- *module load GCC/7.3.0-2.30* *# carga versión específica*

Trabajo con módulos

- ***module list*** ***# ver los módulos cargados***
- ***module unload GCC/8.3.0*** ***# descarga el módulo***
- ***module purge*** ***# descarga todos los módulos***

TIPOS DE TRABAJOS

Tipos de trabajos

- **Serie**
- **En paralelo**
  - **memoria compartida**
  - **memoria distribuida**
  - **híbrido (mem. compartida + mem. distribuida)**
- **GPU (próximamente)**

Tipos de trabajos

- **¿Cómo saber el tipo de trabajo?**

- **Si usted escribió el programa, ya lo sabe**
- **Con aplicación ajena** ⟹ **documentación**

Aplicación	Secuencial	Mem.compartida	Mem. distribuída
Gaussian	x	x	x
Autogrid	x		
Amber	x		x

Tipos de trabajos

- **En conferencia anterior se explicó como trabajar con ellos**

- **Insistir:**

  Trabajos en serie: solicitar 1 cpu

  --nodes=1

  --ntasks-per-node=1

NÚMERO MÁXIMO DE CORES A USAR

¿Cuántos cores máximos usar?

- **Procesadores AMD → 32 cores/nodo**
- **¿Podemos usarlos *todos?***
- **¿Es óptimo usarlos *todos*?**

- **Los procesos del sistema operativo siempre son realizados por los cores**

- **Mejor usar como máximo: 30**

  Dejando 2 cores para los procesos del sistema

SELECCIÓN DE NODOS ESPECÍFICOS

Selección de nodos

- **Hay aplicaciones que requieren especificar el nodo a usar**

- **Ejemplo:**
  En fichero de entrada de Gaussian:
  %LindaWorkers=nodo004

  En el script sbatch:
  #SBATCH --nodelist=nodo004

(Ver con *sinfo* los nodos libres)

CÁLCULO DEL TIEMPO DE EJECUCIÓN

Tiempo de ejecución

- **A veces hace falta determinar el tiempo de ejecución de la aplicación**

- **Hay programas que dan el tiempo en el fichero de salida.**
- **Obtener el tiempo de ejecución con**

  time mi_programa **dentro del script de envío**

ACELERACIÓN Y EFICIENCIA

Cálculo de Aceleración y Eficiencia

- **Cálculo del tiempo: se mide el tiempo *t* *con 1,4,8,16… cores***
- **Cálculo de la Aceleración**

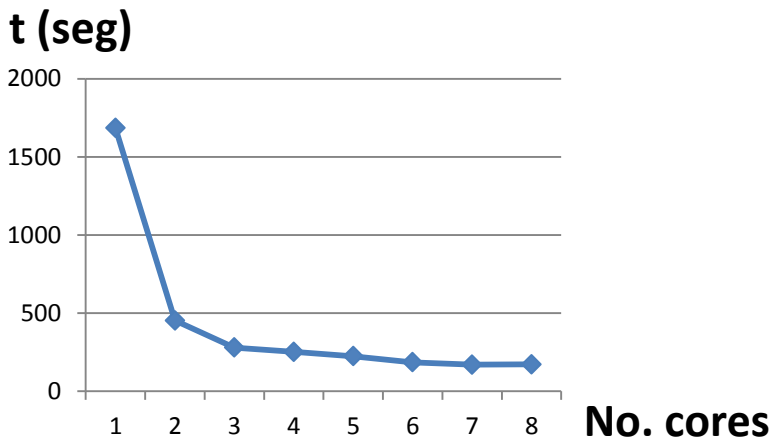
$$A_n = \frac{t_1}{t_n}$$

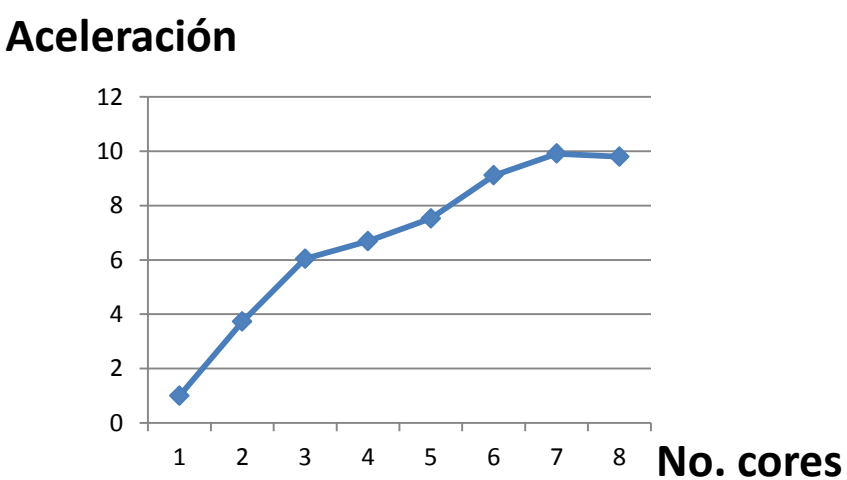
n **= número cores**

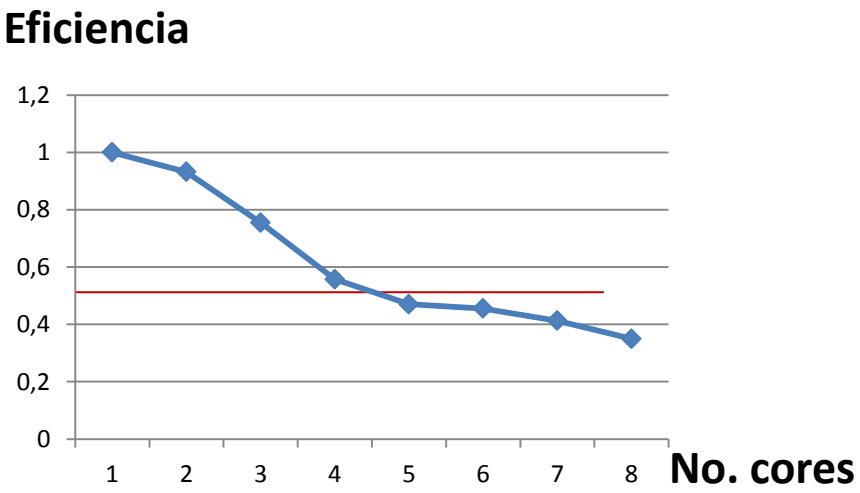
- **Cálculo de la Eficiencia**

$$E_n = \frac{t_1}{n\, t_n}$$

Cálculo de Aceleración y Eficiencia







ENVÍO DE TRABAJOS CON CONDICIÓN

Envío de trabajos con condición

- **A veces hace falta enviar un trabajo con condiciones:**
  - **que se ejecute después que otro finalice OK**
  - **que se ejecute después que otro empiece**
  - **etc**

Envío de trabajos con condición

- **Ejemplo : Que el job 2 comience si el job 1 terminó bien.**

  sbatch job1.sl
  Submitted job 4305

- ***En el script de envío del job2:***
  #SBATCH --dependency=afterok:4305
- ***o también:***
  sbatch --dependency=afterok:4305 job2.sl

Envío de trabajos con condición

- **Otras dependencias:**
- **after - depués que el job haya comenzado**
- **afterany - …. los jobs hayan terminado**
- **afternotok …. el job haya terminado con fallo**

Lista completa de dependencias: *man sbatch*

Envío de trabajos con condición

- ***Pueden ser varios jobs:***
- ***dependency=afterok:4305:4306:4307 job4.sl***

ARREGLOS

Arreglos

- **¿Necesita enviar muchos trabajos similares?**

  - No es óptimo enviar uno a uno
  - Mejor: enviar un arreglo de trabajos

Arreglo

- **múltiples trabajos idénticos**
- **múltiples trabajos que difieren en algún argumento**

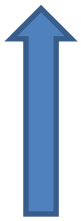
Arreglos

- En vez de N trabajos independientes, es mejor enviar 1 trabajo de arreglo con N tareas
- Millones de tareas - milisegundos
- array job, job array, array task
- Privativo de sbatch (en script o como comando)

Arreglos

Especificación:

- **#SBATCH --array=0-30**
- **#SBATCH --array=1,3,5,7**
- **#SBATCH --array=1-7:2 # con paso=2**
- **#SBATCH --array=0-30%5 # 5 trabajos ejecutándose**



¡Piense en los demás! Use 4% o %5

Arreglos

Variables de ambiente:

- **SLURM_ARRAY_JOB_ID**
  SLURM_ARRAY_TASK_ID

- **Si enviamos un arreglo=1,2 y se le da el job ID 36**
  Submitted batch job 36
  SLURM_JOB_ID=36
  SLURM_ARRAY_JOB_ID=36
  SLURM_ARRAY_TASK_ID=1 (task 36_1)

  SLURM_ARRAY_TASK_ID=2 (task 36_2)

- **Se pueden usar en los scripts sbatch**

Arreglos

Nombres de ficheros (stdin, stdout, stderr):

- **%A sustituye a $SLURM_ARRAY_JOB_ID**

  %a sustituye a $SLURM_ARRAY_TASK_ID

- **#SBATCH --output=array_%A_%a.out**

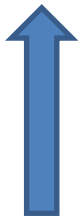
  #SBATCH --error=array_%A_%a.err

Arreglos

Email

- **Se usa #SBATCH --mail-type=END,BEGIN,FAIL**
- **Para especificar el envío por email del arreglo:**

#SBATCH=mail_type=ARRAY_TASKS



¡Cuidado!

Arreglos

Caso 1: 1000 jobs similares, los mismos recursos, ficheros de entrada *diferentes*

Ficheros de entrada: fichero-1, fichero-2 ,….., fichero-1000

- **#SBATCH --array=1-1000%5 # 1000 jobs ejecutándose 5 cada vez**
  #SBATCH --output=array_%A_%a.out
  #SBATCH --error=array_%A_%a.err

  ./programa < fichero-$SLURM_ARRAY_TASK_ID

Arreglos

Ejemplo Caso 1: Gaussian

- **Ficheros de entrada: compuesto-1.com, compuesto-2.com,…**

- **#SBATCH --array=1-1000%5 # 1000 jobs ejecutándose 5 cada vez**

  #SBATCH --output=array_%A_%a.out

  #SBATCH --error=array_%A_%a.err

g09 < compuesto-{$SLURM_ARRAY_TASK_ID}.com > compuesto-{$SLURM_ARRAY_TASK_ID}.log

Arreglos

Caso 2: El mismo anterior pero ficheros en diferentes directorios

- **Directorios: directorio1/, directorio2/ …. con fichero-1, fichero-2,…**
- **Hacer *lista_directorios.txt* con los nombres de los directorios *en columna***

- **#SBATCH --array=1-1000%5 # 1000 jobs ejecutándose 5 cada vez**
  #SBATCH --output=array_%A_%a.out
  #SBATCH --error=array_%A_%a.err

  DIR=$(sed -n "${SLURM_ARRAY_TASK_ID}p" lista_directorios.txt)
  cd $DIR

  ./programa < fichero-$SLURM_ARRAY_TASK_ID

Arreglos

Ejemplo Caso 2: Gaussian

- **Ficheros entrada: compuesto-1.com, compuesto-2.com,…**
  en directorio1, directorio2, …..

- **#SBATCH --array=1-1000%5 # 1000 jobs ejecutándose 5 cada vez**
  #SBATCH --time=01:00:00
  #SBATCH --output=array_%A_%a.out
  #SBATCH --error=array_%A_%a.err

  DIR=$(sed -n "${SLURM_ARRAY_TASK_ID}p" lista_directorios.txt)
  cd $DIR

  g09 < compuesto-{$SLURM_ARRAY_TASK_ID}.com > compuesto-{$SLURM_ARRAY_TASK_ID}.log

Arreglos

Caso 3: El caso 2 pero con 5000 jobs en vez de 1000

Hacer 5000 líneas (jobs) en 1 nodo de una vez es poco eficiente.

Mejor: ciclos con 1000 líneas en 1 nodo (Ahora: 5 jobs haciendo 1000 líneas)

- **#SBATCH --array=1-5**

```
#SBATCH --output=array_%A_%a.out
#SBATCH --error=array_%A_%a.err

NUMLINES=1000
STOP=$((SLURM_ARRAY_TASK_ID*NUMLINES))
START="$(($STOP - $(($NUMLINES - 1))))"
for (( N = $START; N <= $STOP; N++ ))
  do
  DIR=$(sed -n "${SLURM_ARRAY_TASK_ID}p" lista_directorios.txt)
  cd $DIR
  endo

./programa < fichero-$SLURM_ARRAY_TASK_ID
```

Arreglos

Caso 4: Diferentes opciones para un programa

- **#SBATCH --array=0-4**
  #SBATCH --output=array_%A_%a.out
  #SBATCH --error=array_%A_%a.err

  case $SLURM_ARRAY_TASK_ID in
  0) ARGS="-i foo.txt -o foo.out" ;;
  1) ARGS="-i bar.txt -o bar.out" ;;
  2) ARGS="-i ich.txt -o ich.txt" ;;
  3) ARGS="-i pin.txt -o pin.txt" ;;
  4) ARGS="-i lsin.txt -o lsin.txt" ;;
- **esac**

python program.py $ARGS > output-$SLURM_ARRAY_TASK_ID.txt

Arreglos

- **Para programas de Python y R:**

  - hay que cargar los módulos respectivos para que pueda leer el array task id

- **Python**

```
import sys
jobid = sys.getenv('SLURM_ARRAY_TASK_ID')
```

- **R**

```
task_id <- Sys.getenv("SLURM_ARRAY_TASK_ID")
```

TRABAJO CON PYTHON

Trabajo con Python

- **Se recomienda Python 3**

  module load Python/3.6.6-foss-2018b

- **Tiene 380 módulos instalados**

  Para verlos:

  >>> help(¨modules¨)

Trabajo con Python

- **Para instalar otros módulos:**

  Con pip usando opción --user

1. **Upgradear pip**

   python –m pip install –upgrade pip --user

2. **Poner en $HOME/*.bash_profile***

   PATH=$PATH:$HOME/.local/bin:$HOME/bin

Trabajo con Python

- **Instalar/desinstalar nuevos módulos:**

  - instalar:

  python -m pip install keras --user

  python -m pip install theano scikit-learn –user

  python -m pip install theano==1.0.4 * --user

  - desinstalar:

  python –m pip uninstall keras

Trabajo con Python

- **Aclaraciones**

TensorFlow:

Última versión: 2.0

Procesadores AMD del HPC no soportan AVX (requisito a partir de TensorFlow 1.6)

Instalar TensorFlow1.5

pip –m install tensorflow==1.5 --user

Trabajo con Python

- **Aclaraciones**

PyTorch:

Hay que instalar ¨torch¨ y no ¨pytorch¨

pip –m install torch --user

Trabajo con Python

- **Python (Envío de trabajos)**

Secuencial:

```
#!/bin/bash

…

SBATCH --nodes=1

SBATCH --ntasks-per-node=1

…

module load Python/3.6.6-foss-2018b

python python_serie.py
```

Se envía con sbatch python_serie.py

Trabajo con Python

- **Python (Envío de trabajos)**

Memoria compartida:

```
#!/bin/bash

…

SBATCH --nodes=1

SBATCH --cpus-per-task=2

…

module load Python/3.6.6-foss-2018b

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

python python_multihilo.py
```

Se envía con sbatch python_multihilo.py

Trabajo con Python

- **Python (Envío de trabajos)**

Memoria distribuida:

```
#!/bin/bash

…

SBATCH --nodes=2

SBATCH --ntasks-per-node=2

…

module load Python/3.6.6-foss-2018b

python python_paralelo.py
```

Se envía con sbatch python_paralelo.py

Sugerencias para el envío de trabajos en el HPC UO

¡Gracias!

Oscar Au Alvarez , 24 Enero 2020