Patagón

Tutorial 03 - Ejecución de Tareas

1) SLURM *workload manager*

La ejecución de trabajos se realiza mediante la herramienta SLURM, un *workload manager* y *job scheduler* que gestiona la ejecución de trabajos y recursos computacionales según el orden y prioridad. Gracias a SLURM, es posible una convivencia ordenada y justa entre múltiples usuarios sin interrumpirse entre ellos.

Particiones

Existen las siguientes particiones SLURM:

- `IA` (8x GPUs): ideal para trabajos GPU de inteligencia artificial.
- `L40` (3x GPUs, default): por defecto y es para todo tipo de trabajos GPU.
- `RTXPRO` (1x GPU): para trabajos GPU de escala mediana.
- `A4000` (3x GPUs): para trabajos GPU livianos.
- `cpu` (64x CPU cores): trabajos de CPU secuenciales y paralelos.
- El comando `scontrol show partition <PARTICION>` entrega mas información de cada partición.

2) Solicitar recursos *hardware* al ejecutar

Ejecutaremos un job SLURM con el comando `nvidia-smi`, el cual lista las GPUs disponibles para utilizar.

```
➜  ~ srun -p L40 nvidia-smi -L
No devices found.
```

La respuesta `"No devices found.` es el resultado esperado ya que SLURM asume 0 GPUs por defecto. Para solicitar GPUs, es necesario adicionar el parámetro `--gpus=<num>`, donde `<num>` es la cantidad de GPUs a solicitar de la partición. Por ejemplo, si queremos ejecutar `nvidia-smi -L` con una GPU, seria así:

```
➜  ~ srun -p rtx --gpus=1 nvidia-smi -L
GPU 0: NVIDIA L40 (UUID: GPU-2ee80d8a-f11f-1db1-6b83-d17d0b514179)
```

El valor de `<num>` define la cantidad de GPUs a reservar.

Otro ejemplo, ahora pidiendo 2 GPUs

```
➜  ~ srun -p L40 --gpus=2 nvidia-smi -L
GPU 0: NVIDIA L40 (UUID: GPU-2bf13726-6ccf-5c71-282b-7537a29637be)
GPU 1: NVIDIA L40 (UUID: GPU-2ee80d8a-f11f-1db1-6b83-d17d0b514179)
```

Patagón

```
➜  ~ pat resources
+-------------------------------------------------------------+
|          Patagón Resources (in use) v1.20-20250912          |
+------------+--------------+--------------+--------------+
|  Partition |    CPU cores |          RAM |         GPUs |
+------------+--------------+--------------+--------------+
|      A4000 |        0/128 |  192GB/704GB |          0/3 |
|         AI |       11/128 |  228GB/977GB |          3/8 |
|        cpu |        0/128 |  192GB/704GB |          n/a |
|       L40* |        8/128 |  192GB/704GB |          1/3 |
|     RTXPRO |       16/128 |  192GB/704GB |          1/1 |
+------------+--------------+--------------+--------------+
```

Para mas información, `pat --help` .

3) Ejecución con Contenedores (pyxis/enroot)

En Patagón, las dependencias de librerías o software se satisface mediante contenedores. Es decir, al ejecutar un trabajo por SLURM, el usuario indica el contenedor que necesita para ejecutar el trabajo. Este contenedor se descarga automáticamente por el plugin `pyxis` para SLURM y la herramienta `enroot` para ejecutar contenedores. Así, el contenedor se transforma en el nuevo entorno de trabajo del usuario, el cual puede ser modificado a gusto sin afectar a los otros usuarios ni a la configuración nativa del Patagón. Al finalizar la tarea, el contenedor se desmonta automáticamente del nodo de computo por defecto.

Para poder entender mejor la función de los contenedores, ejecutemos `nvcc --version` para consultar la versión del compilador de CUDA `nvcc` .

```
➜  ~ srun -p L40 nvcc --version
slurmstepd: error: execve(): nvcc: No such file or directory
srun: error: nodeGPU02: task 0: Exited with exit code 2
```

Hemos tenido un error como resultado. Esto ocurre porque el nodo de cómputo carece de alguna versión particular de CUDA. Para lograr hacer funcionar este ejemplo, necesitamos indicar un contenedor que incluya CUDA al momento de ejecutar nuestra tarea. Nvidia GPU Cloud (NGC) ofrece diversos contenedores para utilizar, incluyendo el contenedor de CUDA. En este caso, hemos escogido la versión 12.9 de CUDA y la variante `devel` que incluye las herramientas de desarrollo. El comando para descargar el contenedor es (puede tardar algunos minutos):

```
➜  ~ srun -p L40 --container-name=cuda-12.9  --container-image='nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu24.04
pyxis: importing docker image: nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu24.04
pyxis: imported docker image: nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu24.04
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2025 NVIDIA Corporation
Built on Tue_May_27_02:21:03_PDT_2025
Cuda compilation tools, release 12.9, V12.9.86
Build cuda_12.9.r12.9/compiler.36037853_0
```

Para lograr que pyxis/enroot puedan descargar y utilizar automáticamente contenedores de una URL especificada por el usuario, se recomienda configurar sus credenciales. El parámetro `--container-image=<URL>` indica la URL del

Patagón

```
➜  ~ srun -p L40 --container-name=cuda-12.9  -nvcc --version
pyxis: importing docker image: nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu24.04
pyxis: imported docker image: nvcr.io/nvidia/cuda:12.9.1-devel-ubuntu24.04
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2025 NVIDIA Corporation
Built on Tue_May_27_02:21:03_PDT_2025
Cuda compilation tools, release 12.9, V12.9.86
Build cuda_12.9.r12.9/compiler.36037853_0
```

Nota (Julio 2025): Debido a cambios recientes en `pyxis`, para mantener la misma ubicación dentro del job es necesario incluir `--container-workdir=${PWD}` como opción a `srun`. Por ejemplo, si el usuario es `user` y esta actualmente ubicado en `/home/user/`, entonces:

```
➜  ~ pwd
/home/user
➜  ~ srun --container-name=cuda-12.9 pwd
/
➜  ~ srun --container-workdir=${PWD} --container-name=cuda-12.9 pwd
/home/user
```

4) Ejecución de trabajos

Usando comando `srun` (tiempo-real, anclado a la sesión)

El comando `srun` permite a los usuarios ejecutar trabajos para verlos en ejecución de forma directa e interactiva en el terminal. Este modo es síncrono con la sesión, es decir, el terminal queda anclado a la ejecución y progreso del trabajo, y si uno termina su sesión (i.e., se desconecta del Patagón), entonces el trabajo se cancela (a menos que este usando `screen` o algo similar). El comando `srun` sigue la estructura

```
srun  --container-workdir=${PWD}  <contenedor>  <recursos>  <tarea>
```

Ejemplo `srun`

Usaremos un ejemplo de nuestro repositorio *Patagón samples* (https://github.com/patagon-uach/patagon-samples). En particular `patagon-samples/01-custom-programs/03-GPU-single/` el cual multiplica dos matrices. Primero compilaremos.

```
➜  srun --container-workdir=${PWD} --container-name=cuda-12.9 make
nvcc -O3 -arch=sm_80 -lnvidia-ml -Xcompiler -fopenmp main.cu -o prog
➜
```

Y luego ejecutaremos con una GPU (ese ejemplo fue programado para una GPU solamente)

```
➜  03-GPU-single git:(main) ✗ srun -p L40 --container-workdir=${PWD} --container-name=cuda-12.9 --gpus=1 ./p
GPU MATMUL
size: 30720 x 30720
mode 1
Exploring GPUs
    Driver version: 550.127.08
    NUM GPUS = 1
```

Patagón

```
Initializing A and B.......done
matmul shared mem..........done: time: 9.564180 secs
copying result to host.....done
verifying result...........done
```

Usando comando `sbatch` (desacoplado de la sesión)

El comando `sbatch` permite ejecutar tareas por lotes sin la necesidad de mantener la sesión abierta, es decir el usuario puede desconectarse luego de haber lanzado su trabajo con `sbatch`. Para utilizar `sbatch` es necesario escribir un script, típicamente con extensión `*.slurm` (sirve para mantener orden), en el cual se indican los recursos a solicitar, el contenedor, y la tarea en particular. Una tarea `sbatch` puede contener múltiples llamadas a `srun`, y aun seguirá siendo considerada como una sola tarea. Esto ultimo es de gran utilidad cuando necesitamos construir un gran trabajo compuesto de tareas menores. Por ejemplo, una tarea que conste de pre-procesamiento, procesamiento y post-procesamiento podría ser construida con tres llamadas a `srun` dentro de un script `*.slurm`.

Ejemplo `sbatch`

Usemos el mismo ejemplo de *patagon samples* de github (https://github.com/patagon-uach/patagon-samples) y pondremos la compilación y ejecución en el script para `sbatch`. El siguiente script contiene el detalle de la ejecución para `sbatch` (ya existe en el ejemplo del repositorio)

```
#!/bin/bash

# IMPORTANT PARAMS
#SBATCH -p L40                      # Submit to which partition
#SBATCH --gpus=1            # GPU resources, format TYPE:device:quantity

# OTHER PARAMS
#SBATCH -J TUT03-single-GPU          # Name the job
#SBATCH -o TUT03-single-GPU-%j.out   # Write the standard output to file named 'jMPItest-<job_number>.out'
#SBATCH -e TUT03-single-GPU-%j.err   # Write the standard error to file named 'jMPItest-<job_number>.err'

# COMMANDS ON THE COMPUTE NODE
pwd                         # prints current working directory
date                        # prints the date and time

# compile the GPU program
srun --container-workdir=${PWD} --container-name=cuda-12.9 make
# run the GPU program
srun --container-workdir=${PWD} --container-name=cuda-12.9 ./prog 0 $((1024*40)) 1
```

Ejecutamos el job con `sbatch`:

```
➜  03-GPU-single git:(main) sbatch launch.slurm
Submitted batch job 68448
```

Revisando rápidamente la cola de trabajo con el comando `squeue`, podemos identificar el job por su ID:

```
➜  03-GPU-single git:(main) squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

Patagón

```
             68398       L40 Qwen-Eva     cris  R   12:39:00      1 nodeGPU02
             68416       cpu hy_nnt11  hgarces  R    2:44:17      1 nodeGPU02
             68403       cpu     bash     naye  R    4:12:30      1 nodeGPU02
```

Un job por `sbatch` no genera output en la pantalla, sino que en sus archivos de error (`err`) y salida (`out`).

```
➜  03-GPU-single git:(main) ls
launch.slurm  main.cu  Makefile  prog  srun.txt  TUT03-single-GPU-68448.err  TUT03-single-GPU-68448.out
```

Revisando cada uno podemos ver que no hay errores y que el output es el mismo de cuando se ejecuto con `srun` :

```
➜  03-GPU-single git:(main) cat TUT03-single-GPU-68448.err
➜  03-GPU-single git:(main) cat TUT03-single-GPU-68448.out
/home/cnavarro/patagon-samples/01-custom-programs/03-GPU-single
Thu Jul 17 11:39:38 PM -04 2025
nvcc -O3 -arch=sm_80 -lnvidia-ml -Xcompiler -fopenmp main.cu -o prog
GPU MATMUL
size: 40960 x 40960
mode 1
Exploring GPUs
    Driver version: 550.127.08
    NUM GPUS = 1
    Listing devices:
        GPU0 NVIDIA L40, index=0, UUID=GPU-2bf13726-6ccf-5c71-282b-7537a29637be  -> util = 0%
Choosing GPU 0
initializing A and B.......done
➜
```

Universidad Austral de Chile