Patagón

Entrenamiento de una Red Neuronal Pytorch en el supercomputador Patagón.

Introducción

En este tutorial enseñaremos un caso de uso típico de **Machine Learning** utilizando **Python**. El objetivo es hacer más accesible el uso del supercomputador **Patagón** y (ojala!) despejar dudas que puedan haber quedado luego de repasar el manual de usuario al ver un ejemplo concreto. Este tutorial hará uso de la información entregada en las distintas entradas al manual de usuario del supercomputador **Patagón de la UACh**, las cuales están disponibles en patagon.uach.cl.

La librería a utilizar en este caso en particular es **Pytorch**, sin embargo, el proceso se puede adaptar para utilizar cualquier otra librería.

Una vista general del tutorial sera:

1. Administración del contenedor **Docker**
2. Instalación de librerías y paquetes necesarios
3. Ejecución de la tarea utilizando **SLURM**

Como requisito a este tutorial se requiere de acceso al **Supercomputador Patagón** y haber clonado el repositorio de muestras **Patagon-Samples** disponible en **Github**. Para ello basta con ejecutar:

```
$ git clone https://github.com/patagon-uach/patagon-samples
```

1. Administración del contenedor Docker

El supercomputador Patagón consiste principalmente de 3 nodos conocidos como **Patagon-Master** (**nodo maestro**), **storage01** (**nodo de storage**) y **nodeGPU01** (**nodo de cómputo**). Cada uno tiene un objetivo distinto y es de suma importancia respetar el uso que deben tener. Para los usuarios el nodo **storage** es invisible pero pueden saber que es el encargado que los datos de sus `home` se vean transparentemente sin importar en que sistema están parados. Por lo tanto, las siguientes explicaciones se enfocan mas en los nodos **maestro** y **computo**.

Patagón

lanzar tareas de cómputo en el nodo maestro (ej: programas como `./simulacionUltraCostosa` o `./entrenarMiRed.py` directamente en el nodo **maestro** esta prohibido). En el nodo maestro solo se ejecutan comandos básicos como por ejemplo los de manejo de directorios/archivos (`cat` , `mkdir` , `ls` , `cd` , `vim` , `nano` , etc...) o también para lanzar trabajos SLURM al nodo de cómputo. Para mayor inflacionario sobre la arquitectura del **Patagon** ver aqui.

El **nodo de cómputo** es el que hace el procesamiento en el **Patagon**, por lo que todas nuestras tareas de investigación deben ejecutarse ahí, enviadas desde el **nodo maestro**. El **nodo de cómputo** funciona en base a contenedores **Docker** (internamente utilizando las herramientas **Pyxis** y **Enroot**), lo que nos permite mantener múltiples ambientes **privados** por usuario sin la necesidad de instalar librerías a nivel de sistema, ni contactar al administrador para instalaciones especiales. Con esta forma de funcionamiento, los usuarios se auto-atienden, esto incluye instalar librerías y programas que pueda necesitar. Para más información revisar acá.

2. Descargar contenedor e instalación de paquetes necesarios

Para descargar un contenedor podemos ingresar a https://hub.docker.com donde se encuentran muchos contenedores con distintos paquetes pre-instalados, lo cual nos servirá como un punto de partida. En nuestro caso, dado que nos interesa utilizar **Python** con la libreria **Pytorch** podríamos buscar un contenedor que ya tenga **Pytorch** instalado, sin embargo, para efectos prácticos utilizaremos un contenedor de Python estándar:

https://hub.docker.com/_/python

Importante: Es posible que para descargar el contenedor que necesites debas seguir los pasos de esta entrada al manual.

El formato para ejecutar cualquier instrucción dentro de nuestro contenedor `python3.8` en la partición `cpu` seria:

```
$ srun --container-workdir=${PWD} --container-name=python3.8 --container-image='python:3.8' -p cpu --pty <in
```

Acá **container-name** es un nombre arbitrario, decidido por el usuario, para identificar al contenedor de los demás que hayas descargado y **container-image** es el nombre del contenedor de **Docker Hub**. En este caso especificamos la versión de Python que requiero dentro del contenedor utilizando `:` . Con `-p cpu` especifica que queremos ejecutar nuestra tarea en la partición SLURM `cpu` del nodo de computo, ya que no se necesitan GPUs. El nodo de computo se divide en dos particiones `gpu` y `cpu` , para un mayor detalle del funcionamiento y los recursos de partición una ver acá.

Comencemos con ejecutar `bash` para abrir una sesión interactiva en el contenedor.

```
$ srun --container-workdir=${PWD} --container-name=python3.8 --container-image='python:3.8' -p cpu --pty bas
```

En respuesta al comando anterior veremos como se descarga el contenedor:

```
pyxis: importing docker image ...
```

Patagón

```
user@nodeGPU01:~$
```

Es importante destacar que cada vez que se ejecute el comando `srun` en el **nodo maestro** la instruccion que le sigue se llevara a cabo dentro del **nodo de cómputo**, en el comando anterior, al estar ejecutando `bash` se están reservando **recursos** del **nodo de cómputo** y abriendo una sesión **interactiva**. Es de suma importancia mantener el ingreso **interactivo** al **nodo de cómputo** al mínimo para no desperdiciar los limitados recursos y utilizarlo solo para instalar paquetes.

En cualquier momento podemos usar el comando `exit` para salir del contenedor (nodo de computo), liberar los recursos pedidos y regresar al nodo maestro. Por ahora nos quedaremos en la sesión interactiva para instalar la librería que necesitamos en nuestro contenedor. Afortunadamente, utilizando `pip` uno puede instalar liberias a nivel de usuario sin necesidad de privilegios root. Instalaremos **PyTorch** usando:

```
$ python -m pip install torch==1.11.0+cu113 torchvision==0.12.0+cu113 torchaudio==0.11.0+cu113 -f https://do
```

*: comando de instalación obtenido de https://pytorch.org/get-started/locally/ Cuando finalice la instalación podemos ingresar a una sesión interactiva de **Python** usando `python` y probar con `import torch`.

```
user@nodeGPU01:~$ python
Python 3.8.12 (default, Mar  2 2022, 04:56:27)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>>
```

0 errores! En este punto estaríamos dentro de una sesión `ssh` en el nodo maestro, la cual mantiene una sesión interactiva dentro del nodo de cómputo, dentro del contenedor la que está ejecutando python interactivamente (inception!). A continuación podemos salir de la sesión interactiva de **Python** con `CTRL+D` o usando `exit`.

Finalizamos la sesión interactiva `bash` con `exit`. Una vez de vuelta en el **nodo maestro**, ingresamos otra vez al contenedor con el flag `--container-remap-root`

```
$ srun --container-workdir=${PWD} --container-name=python3.8 --container-image='python:3.8' -p cpu --contain
```

*: acá `srun` es una instruccion de **SLURM**, el gestor de trabajos que utiliza el **Supercomputador Patagón**. Para más información, ver https://slurm.schedmd.com.

De esta manera ingresamos al contenedor como root, lo que nos permitira instalar paquetes a nivel de sistema en nuestro ambiente del contenedor. En este caso particular, el ambiente que viene con el contenedor docker está basado en ubuntu, por lo que el gestor de paquetes es `apt`. Para instalar un paquete bastaría con ejecutar:

```
$ apt update
$ apt install <nombre del paquete>
```

En caso de que CUDA no venga por defecto con pytorch, podemos instalarlo nosotros mismos en nuestro contenedor

```
apt install nvidia-cuda-toolkit
```

Patagón

Supercomputador Patagon utiliza contenedores, cada usuario ya tiene su ambiente privado con acceso `root`, lo que podría servir para instalar librerías que se requieran a nivel de sistema, una de las características más útiles de `conda`.

3. Ejecucion de la tarea utilizando **SLURM**

De vuelta en el **nodo maestro** con nuestro contenedor preparado para cualquier tarea que requiera **Pytorch**, procederemos a crear un script de **SLURM**, con toda la información necesaria para la ejecucion de nuestra tarea.

SLURM es el gestor de colas que utiliza el **Patagón**, este programa se encarga de organizar todos los trabajos que se quieran realizar al proporcionar orden de ejecucion y administracion de los recursos

MiTareaML.slurm

```
#!/bin/bash

# IMPORTANT PARAMS
#SBATCH -p IA                       # Particion GPU
#SBATCH --gpus=1           # Una GPU por favor

# OTHER PARAMS
#SBATCH -J MiTareaML          # Nombre de la tarea
#SBATCH -o MiTareaML-%j.out   #
#SBATCH -e MiTareaML-%j.err   #

# COMMANDS ON THE COMPUTE NODE
pwd                          #
date                         #

# Ejecutar nuestro programa
cd patagon-samples/05-Tutoriales                    # navegacion al
cd 01-machine-learning-pytorch                      # directorio
srun --container-workdir=${PWD} --container-name=python3.8 python main.py
```

*: Para mayor información sobre el script y el funcionamiento de **SLURM** ver acá.

Por ahora dejaremos este **MiTareaML.slurm** en el directorio raiz de nuestro usuario. \ Finalmente ejecutaremos

```
$ sbatch MiTareaML.slurm
Submitted batch job 20471
```

para manadr nuestro **job** a la cola de **SLURM**.

Haciendo `cat MiTareaML-<jobid>.out` podemos monitorear la salida de nuestro **job**:

```
$ cat MiTareaML-20471.out
/home/user
Sat 12 Mar 2022 10:49:32 PM -03
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.329474
Train Epoch: 1 [640/60000 (1%)] Loss: 1.425025
Train Epoch: 1 [1280/60000 (2%)]    Loss: 0.797880
```

Patagón

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
20195       AI  python3 useruser  R 6-11:02:57      1 nodeGPU01
20299       AI importan    user2  R 3-09:53:03      1 nodeGPU01
20471       AI MiTareaM     user  R       0:04      1 nodeGPU01
```

Podemos usar `scancel` para cancelar la ejecucion de nuestro **job**:

```
$ scancel 20471
```

Si somos afortunad@s lo veremos en ejecucion inmediatamente. En caso contrario, se encolara hasta que haya recursos disponibles para su ejecucion. \ De cualquier manera, ahora podemos cerrar tranquilamente la sesión `ssh` sin miedo a que se cancele nuestro **job**, **SLURM** se encargará del resto.

Con esto damos por concluido el primer tutorial sobre el uso del **Supercomputador Patagon de la UACh**, gracias por leer y les deseamos mucho éxito en sus investigaciones.

Cualquier comentario sobre este documento será agradecido. Lo pueden hacer llegar al correo del **Patagón** patagon@uach.cl.

Universidad Austral de Chile