GETTING STARTED IN COMPUTE CANADA

Andre Pacheco and Abder-Rahman Ali
Laboratory for Hierarchical Anticipatory Learning - HALLab
Dalhousie University, Halifax, NS, Canada

Introduction

What's Compute Canada?

Compute Canada is the national advanced research computing (ARC) facility of Canada. It's a not-for-profit corporation and a federation of 37 member universities and research institutions. In a brief, it's a share computer infrastructure that serves more than 10,000 researchers including more than 3,000 faculty across the country. It provides access to powerful computers and GPUs using elastic compute cloud.

What's the purpose of this tutorial?

Getting started in Compute Canada may be a little bit tiresome. There are too many pieces of information in the official guide and it may be tricky to learn how to use it in a fast way. So, the main goal of this tutorial is to teach you the basics of this environment. It doesn't intend to replace the official guide (at all), you'll still need to check it sometimes. However, it's a summary that will quickly introduce you to the Compute Canada ecosystem. In summary, we just want to save your time. Instead of spending a lot of time trying to understand how it works, you can just read this tutorial and start to use the system.

Getting the hands dirty

Registration

First of all, you need to register yourself. Go to Compute Canada login page and, next to the sign in button, click in register. After that, you need to agree with all terms (of course you'll read all of them) and answer if you have ever applied for an account. In the following, you need to put your personal information. Obviously, you need to use your dal.ca in the e-mail box. After you insert all pieces of information about you, **they will ask for the sponsor/supervisor number**. You need to ask this code for your sponsor/supervisor. Since you get this number, put it in the box and submit your application. They'll check your information and send an e-mail to your sponsor/supervisor. As soon as he/she confirms that you're who you're, the system will send an e-mail for you saying you're fine to use it.

Connecting into the servers

As stated before, Compute Canada works in the cloud. Thereby, you're going to need to connect to a server in order to run your models. Once your application was approved, you're going to have an account with your user name and password that you have put on the registration form. The platform has different servers names, such as `graham` and `beluga`. You may check all of them in Compute Canada Status. We just need to choose one server name and connect using `ssh` protocol. If you don't know how to use this protocol, don't worry, we're going to teach you the basics. In summary, we have two options here: 1) we can use an `ssh` client software for Windows, MacOS or even for Linux. It means you're going to have a graphic interface to use it. 2) or we can use a shell terminal combined with some commands. In this tutorial we're going use **Linux terminal**, but the same commands are valid to MacOS.

The example below connects the user goku to the server graham. The $ character means we're using the shell terminal.

```
    $ ssh goku@graham.computecanada.ca
```

After type the above command and press enter, a message will show up on the screen asking for your password. Just put it there and you'll be connected to the system. It's worthy to note that if the graham server is busy, you can choose another one to connect. For now on, we'll always use the user goku and the server graham as examples, but remember you can change that, it's just for simplicity.

Pro tip: If you're using Linux, you can configure ssh to connect to the server typing a little bit less. Go to ~/.ssh and open the config file. Next, create a file exactly as follows:

```
Host graham
    User goku
    HostName graham.computecanada.ca
```

Now, to connect to the server, just type on terminal: `ssh graham`.

Accessing my user folder

After getting logged in the server you need to go to your account folder. If you type ls on terminal, you're going to see some folders. Using the command cd, you must open the project folder. Inside this folder, you're going to see the manager user folder. This folder has all accounts related to your supervision/sponsor. Now, open this folder and you're going to see different folders with different user names. Your folder will be there, for example, goku. For security reasons, you don't have access to the remaining folders, only yours. In summary, the path you should use will be always something like projects/sponsor_username/username. Everything you want to send to the server must be addressed to this folder.
Pro tip: to know the full path of your folder use the command pwd. You're going to get something like /home/username/projects/_username/username. It's important to transfer data!

Transferring data to your folder

To run your models you're going to need to send data to the server. There are different protocols and softwares you can use to achieve this goal. We're going to teach one way and recommend a tutorial for another.
A simple way to transfer data is using the scp. It's similar to ssh, but it send and receive data to/from servers. To send/receive data using it you need to use the following command pattern:

```
$ scp [-r] <source-path> <destination-path>
```

The <source-path> is where the file(s) you want to send are and the <destination-path> is where you want to put them. If you want to send an entire folder, use the parameter -r, otherwise, you don't need to use it. Suppose we want to send the folder dataset, which is in /home/goku/dataset to the server graham. As we know, we need to put it in our account folder. Nonetheless, we need to use the full path to this folder. So, we're going use the following path: /home/goku/projects/sponsor_username/goku. Now, we just need to type the following command on terminal:

```
$ scp -r /home/goku/dataset goku@graham.computecanada.ca:/home/goku/projects/sponsor_username/gok
```

Next, the protocol will request your password and, if you type it correctly, the transference will start. If you'd like to transfer data from server to your machine, you just need to change source and destination. In addition, if you'd like to transfer data between two servers, just use goku@server1.ca:/path and goku@server2.ca:/path.

If you don't like to use terminal, you can use the `sftp` protocol using a graphic software like FileZilla. You just need to install it and check this tutorial to learn how to use it.

Setting up the environment

Modules

In order to run your models, you need to set up the environment. To install packages in your account, you're going to use a special package manager called `module`. It's used to load packages that are already installed in the server to your account. So, to check everything you can do with this command you can use `module help` or check the official documentation.

In the following, we present the most useful module commands. To use them, just type the command on terminal that is logged in your account.

- `$ module avail`: it shows all modules available to be load in your account. Alternatively, you can check it here.
- `$ module avail <module_name>`: it's just to reduce the amount of results of the previous command. So, you can type the module's name you are looking for and check if it's in the list that will be shown on the screen.
- `$module load <module_name>`: this is the most important command. It loads the module to your account. So, suppose you want to "install" Python in your account. Thus, you can use the command `module load python/3.6`. Now, if you type `python3` on terminal, it's going to work. **However, it loads a given module only for the current session**. It means the next time you login in your account the module won't be reloaded. To make it load every time you login in your account, you need to include this command line in `/.bashrc`.
- `$ module unload <module_name>`: As the name suggests, it unloads a given module for the current session.

Python and the virtual environment

If you're using Python, the best option to set up your environment is to use a **virtual environment**. In summary, you install your packages using `pip` inside the virtual environment and just activate it when you want to use it. To build a virtual environment in Python you need to use the `virtualenv` command. Next, we present a step-by-step to how to get it:

1. **Installing**: to install the `virtualenv` type the following command on terminal:

   ```
   $ python -m pip install --user virtualenv
   ```

2. **Creating the virtualenv**: to create the virtual environment do the following:

   ```
   $ python -m venv <NAME>
   ```

   The `<NAME>` is just a alias you create to your `virtualenv`. Let's name this as *gandalf*.

3. **Activating the virtualenv**: now you installed it, you need to activate the venv as follows:

   ```
   $ source gandalf/bin/activate
   ```

   Once you activated it, you're going to see something like: `(gandalf) [goku@graham] $`. It means you're using the `virtualenv` and you can use `pip` to install the all packages you need. Now, every time you log in your account, you just need to activate the `virtualenv` and run your code inside it.

4. **Deactivating the virtualenv**: to deactivate the `virtualenv` you just need to use the following command:

```
$ deactivate
```

Working with jobs

What's a job?

In a brief, a job is any piece of code you'd like to run in the clusters. In traditional computers, you can use windows, menus, buttons, etc. in order to run a given task. On Compute Canada you need to use the command line interface to run your jobs. However, as it's a shared environment, the resource may not be available for you when you want to run your code. Therefore, you're going to put your job in a waiting list. To do so, you need to prepare a small text file called **job script** that basically says what program to run, how many resources you need, among many others configurations. After that, you must submit this script to a job scheduler which decides when and where it will run. Basically, when the requested resources are available, your job will run. So, it works like a referee and prevents the interference among different jobs.

The job scheduler

The job scheduler is a piece of software with multiple responsibilities, which includes:

- Maintain a database of jobs
- Enforce policies regarding limits and priorities
- Ensure resources are not overloaded, for example, by only assigning each CPU core to one job at a time
- Decide which jobs to run and on which compute nodes
- Launch them on those nodes
- Clean up after each job finishes

On Compute Canada clusters, these responsibilities are handled by the Slurm Workload Manager. As you already know, here we're going to present the basics and if you're needing to do something that is not covered in this tutorial, you should refer to the `Slurm` website.

Running jobs

Now you know what's a job, let's start to run them. As you know, we need to prepare a script and submit it to the job scheduler. Our first script is described below:

```
#!/bin/bash
#SBATCH --time=00:01:00
#SBATCH --account=sponsor_username
echo 'Hello, world!'
sleep 30
```

If you're not familiar with shell script, don't worry. We're going to describe all you need to know. The 1st line is used just to inform to the system that it's a shell script. Everything that starts with `#SBATCH` is a command for `Slurm`. In this example, in the 2nd line, we describe how much time our job will need. This is important! If you request less time than you need, your job will be closed before it's done. In the 3rd line, we declare our account group name, which is the sponsor/supervisor username.

Any job you submit needs to have it. You can find it's information on Compute Canada website or just looking into `/home/goku/projects`. The folder that shows up in this directory has the same name that your group account. Finally, in the 4th and 5th lines are the code, which is your job, to be executed. Of course, it's a pretty simple job, but you can call your code in these lines, for example, `python do_something.py`. Once your script is done, you need to submit it to the scheduler as follows:

```
$ sbatch simple_job.sh
Submitted batch job 123456
```

The reserved word `sbatch` is the command you need to call for the job scheduler. Now, as soon as possible, your job will be executed. In this example, it's almost on the fly, since it's pretty simple. After its execution, the software produces a log file named like `slurm-123456.out`, which 1234456 is the ID generated for your job. Everything that is printed on the screen during your code execution, will be stored in this file. Actually, you can change its name, only if you want to, using the command `#SBATCH --output=new_name.out`.

My job script

Now, you already know that we need a job script to schedule our job. In this section, let's write a very useful script to run our jobs. This script is described below:

```
#!/bin/bash
#SBATCH --account=sponsor_username
#SBATCH --time=20:00:00
#SBATCH --mem=100G
#SBATCH --gres=gpu:2
#SBATCH --cpus-per-task=16
#SBATCH --mail-user=goku@dal.ca
#SBATCH --mail-type=ALL

python my_code.py
```

The first two lines were explained in the last section. The remaining commands are described in the following:

- `--mem=100G`: it represents the amount of memory you're going to need in order to run your job. It's very important to set enough memory, otherwise, you're job will be aborted.
- `--gres=gpu:2`: it asks for 2 GPUs to run your job. Some servers, such as `graham`, you can ask for a specific type of card, for example, `--gres=gpu:v100:2`, which is requesting 2 Nvidias V100. You can check the amount of resource available for each server in the official manual.
- `--cpus-per-task=16`: similar to the previous one, but, for CPUs.
- `--mail-user=goku@dal.ca` and `--mail-type=ALL`: these two commands are used to send an email to the given email address when something happens with your job. As we're using ALL, `Slurm` will warn you when the job start, finish, if it failed etc. You can control how much information you'd like to have changing ALL for END, for example, but we strong recommend you to use ALL, otherwise, if your job fails, you won't informed by that. Believe us, it's very useful!

This script covers the basic configuration that you need in order to run your model in a GPU. Again, if you'd like to configure even more, have a look on Slurm manual. They have all commands documented there.

Checking my jobs status

After scheduling your job, you may want to check its status. There are different status that your job may assume, the most common ones are:

- **Pending (PD):** Job is waiting for resource allocation
- **Running (R):** Job currently has an allocation and is running
- **Failed (F):** Job terminated with non-zero exit code or other failure condition
- **Completed (CD):** Job has terminated all processes on all nodes with an exit code of zero
- **Timeout (TO):** Job terminated upon reaching its time limit

In order to get this status you need to use the following command:

```
$ squeue -u $USER
```

The `$USER` is used to get your username, but you can type it here if you want so. Beyond the status, you can get the job ID, the remaining time to get a TO etc. To get more detail about a finished job, you may use its ID and the following command:

```
$ seff 12345678
Job ID: 12345678
Cluster: cedar
User/Group: jsmith/jsmith
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 02:48:58
CPU Efficiency: 99.72% of 02:49:26 core-walltime
Job Wall-clock time: 02:49:26
Memory Utilized: 213.85 MB
Memory Efficiency: 0.17% of 125.00 GB
```

It's going to show you how long it took, the amount of memory used, among others pieces of information.

Cancelling a job

If you scheduled a job using `sbatch` and now you want to cancel it, you can use the following command:

```
$ scancel <JOB_ID>
```

The `<JOB_ID>` is obtained by `squeue`. You can also cancel jobs in batches, for example:

```
$ scancel -u $USER
$ scancel -t PENDING -u $USER
```

In the previous example, the first line cancel all jobs of the current user and the second one cancel all pending jobs. You can insert another status instead of pending.

Interactive jobs

Sometimes you just want to make a quick test and track the results on the terminal screen to debug something, for example. You can do that using the command `salloc`:

```
   salloc --time=03:00:00 --gres=gpu:2 --cpus-per-task=16 --account=def-ttt --mem=100000M
```

Observe that the parameters are the same we used in the scripts. After running this command, the scheduler will check if these resources are available. If yes, it'll redirect you to a terminal and you can run your code as usual. If not, you'll need to wait. Notice that if you're running a job using the interactive approach and close your terminal, the execution will be interrupted. This is a disadvantage of this method.

Final tips

- **Make sure your environment is set up when you schedule your job**. If your job need a package or lib that is not available in the environment that you're trying to run, it's going to fail. If you're using a **virtualenv**, for example, **make sure to activate it before you schedule the job**.
- To certify the first tip, try to use an interactive job just to certify your code is fine. Next, you can schedule it. It may save some time for you if you get a error.
- By the way, if your job was running and it failed, check the error code in slurm output. It's an important clue to help you figure out what happened. Here you can see the most common code errors.
- Always use a logger in your code. If you're using Python, try the `logging` package.
- If you're training a model, save the checkpoints! If your job runs out the time or failed for lack of memory resource, you're going to be safe if you have the checkpoints.