Running Algorithms Locally and Remotely via Docker

Your computing committee

Contents

1 Introduction

This document explains the current software used to run locally and remotely algorithms inside Docker. The focus is on explaining the salient points of different scripts so that you can modify them to suit your algorithm. We won't explain the technicalities involved in using `docker`, `tmux` and `ssh`. For a basic introduction to those tools, the reader is referred to the wiki maintained by the statistics department at stat.cmu.edu/computewiki.

The example project is a simplified version of Sparse Information Filter for Fast Gaussian Process Regression (Kania et al. 2021), which proposes a method for training sparse Gaussian processes via mini-batches rather than full-batches.

The structure of the folder that contains this file is as follows

- `Dockerfile` configures a docker image. At a basic level, it is a declaration of the environment where the algorithm will run.
- `DockerfileGPU` configures a docker image with GPU support.
- `.dockerignore` configures exceptions when creating the docker image
- `rundocker.sh` runs the algorithm inside the container.
- `src` contains the source code of the algorithm
- `data` contains the data used for the experiments
- `results` is the folder where the results of the experiments are saved
- `utilities` contains useful scripts for remotely running algorithms
- `README` contains the files needed to generate this document
- `paper.pdf` The paper whose source code is used in this project. The algorithms implemented inside the `src` folder are explained in it. `supp.pdf` contains additional information.

Many of the code snippets shown in this document have been substantially simplified to reflect the essence of the scripts. However, the scripts in the root folder differ since they target a specific algorithm rather than a generic one.

2 Running docker locally in your computer

2.1 Install docker

Get the latest version of docker desktop from docs.docker.com/desktop. Once the installation finishes, please start docker desktop.

2.2 Configure a docker image

`Dockerfile` configures the docker image, that is, the environment containing all the dependencies necessary for running your algorithm.

We start by declaring a base image on which we will install further dependencies. For example, the following code provides a `linux` system with `python` 3.8 support.

```
FROM python:3.8-slim-buster
```

Consult the `Dockerfile` for further instructions on how to use a base image with `Pytorch` and `Tensroflow` with GPU support.

On it, we might install further dependencies via `pip`. It is recommended to specify the version of the required dependencies. Otherwise, `pip` will install the latest version compatible with our other packages making the docker image non-deterministic.

```
# We install additional packages required for this particular project
RUN pip install pandas==2.0.3
```

Then, we copy the code required to run the algorithm and experiments

```
# We copy the folder containing the source code of our algorithm
# into the folder /program/src in the container
COPY src /program/src
```

We do not copy the data or results folders into the container. They will be mounted to the Docker container during runtime.

2.3 Building a docker container

Our next step is to start a Docker container based on our image. The script `rundocker.sh` simplifies the building procedure. To wit, it

- creates a docker container based on the image defined in `Dockerfile`
- mounts the data folder instead of copying it into the container. This avoids making images unnecessarily large due to large datasets.
- mounts the results folder instead of copying it into the container. Therefore, if the computer crashes, the saved data will remain after reboot.

- runs an algorithm inside the container.

The most important part of the script is indicating the location of your algorithm's executable. Everything else in the file might be left untouched if you follow the conventions adopted by this tutorial.

```
#####################################################
# Location of the executable inside the container
#####################################################
readonly executable=/program/experiments/runner.py
```

2.4 Running an algorithm in the docker container

To run your container, execute the script `rundocker.sh`. You will immediately see the output of our algorithm.

```
bash rundocker.sh
```

If you wish to pass arguments to your executable, you can do so by running

```
bash rundocker.sh your_arguments
```

Then, you can then parse `your_arguments` inside your script.

3 Remotely running one instance of the algorithm

3.1 Configure the server

A ssh connection to the server can be configured at /.ssh/config. For example, using the following code, you can configure access to hydra1.stat.cmu.edu.

```
Host hydra1
  Hostname hydra1.stat.cmu.edu
  Port 22
  user your_usename
  IdentityFile ~/.ssh/your_public_key_file
  ServerAliveInterval 180
```

Thus, we execute

```
ssh hydra1
```

in the terminal, ssh will automatically connect to hydra1.stat.cmu.edu using the public key ~/.ssh/your_public_key_file.

3.2 Copying your files to the remote server

We can copy all the files in this folder to hydra1 using scp

```
scp -r $PWD hydra1:~/remote/sparsegp
```

3.3 Remote execution and monitoring

After running the command in the previous section, rundocker.sh will be located at /remote/rundocker. You can use the utility distribute.sh to connect to the remote server and run your rundocker.sh script.

```
bash distribute.sh --server hydra1 --session TEST --rundocker "~/remote/
    sparsegp"
```

distribute.sh connects to the hydra1 server, and runs the rundocker.sh script inside a tmux session called TEST.

To observe the output of our algorithm, we can use the process.sh utility. Specify the tmux session name and the server as follows.

```
bash process.sh --session TEST --server hydra1
```

4 Remotely running multiple instances of your algorithm

You can run one instance per server of your algorithm with different parameters in each server. This is useful if you run the same experiments multiple times with different parameters.

4.1 Configure the servers

We configure four servers at /.ssh/config, two `ghidorahs` and two `hydras`.

```
Host ghidorah2
  Hostname ghidorah2.stat.cmu.edu
  Port 22
  user your_usename
  IdentityFile ~/.ssh/your_public_key_file
  ServerAliveInterval 180
Host ghidorah3
  Hostname ghidorah3.stat.cmu.edu
  Port 22
  user your_usename
  IdentityFile ~/.ssh/your_public_key_file
  ServerAliveInterval 180
Host hydra1
  Hostname hydra1.stat.cmu.edu
  Port 22
  user your_usename
  IdentityFile ~/.ssh/your_public_key_file
  ServerAliveInterval 180
Host hydra2
  Hostname hydra2.stat.cmu.edu
  Port 22
  user your_usename
  IdentityFile ~/.ssh/your_public_key_file
  ServerAliveInterval 180
```

Subsequently, we specify the servers where you want to run the instances of your algorithm in `servers.sh`

```
# Declare servers to be used for running your scripts
servers=(
 ghidorah2
 ghidorah3
 hydra1
 hydra2
)
```

4.2 Remote execution and monitoring

The process is analogous to running one instance, but we must iterate over the declared servers in **server.sh**. The utility **experiments.sh** gives one example of how to achieve this. In short, we load the server configuration by executing

```
##############################################
# load servers
##############################################
. servers.sh
```

Then, we declare an array with the parameters that we want to pass to the algorithm

```
##############################################
# Declare parameters that change from server to server
##############################################
parameters=(
 param_for_ghidorah2
 param_for_ghidorah3
 param_for_hydra1
 param_for_hydra2
)
```

and iterate over the servers

```
##############################################
# We iterate over the servers and run the script
# with different parameters in each one of them
##############################################
for i in "${!parameters[@]}"; do
  ./distribute.sh --server "${servers[i]}"    \
  --session PARAMETERS                \
  --rundocker "~/remote/"             \
  "${parameters[i]}"                  \
done
```

We can run the script via bash

```
bash experiments.sh
```

To monitor the four instances, we can execute

```
bash process.sh --session TEST
```

Note that we specify the session but not the server. The utility **process.sh** will iterate over the servers declared in **servers.sh** and connect to each session.

4.3 Resetting everything

If you want to stop the container in all the servers specified in **servers.sh**, execute

```
bash clean.sh
```