Data-intensive Computing - Assignment 3

Object Detection

Benjamin Lee (e12112693)
Lukasz Sobocinski (e12123563)
Valentin Milicevic (e12122084)

June 23, 2022

1 Datasets

The model for object detection is a pre-trained saved model from the Tensorflow Hub repository, which models can be reused to solve new tasks. The pre-training stage helps the model detect an object on the image fluently.

There are four datasets provided on TUWEL for object detection [1]. The first dataset is the original dataset, and three others are just resized versions of the original dataset.

- Test2017.zip
  - Number of images: 40,671
  - Size: 6.64 GB
- Object-detection-BIG.zip
  - Number of images: 4,914
  - Size: 763 MB
- Object-detection-MEDIUM.zip
  - Number of images: 2,071
  - Size: 316 MB
- Object-detection-SMALL.zip
  - Number of images: 297
  - Size: 45.5 MB

2 Methodology

2.1 Experiment Setup

Here are listed key specifications from both the local machine used for local execution and the cluster machine used for remote execution.

- Local Machine
  - Processor: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
  - RAM: 16.0 GB (15.7 GB usable)
  - System type: 64-bit operating system, x64-based processor
- Cluster Machine (Little Big Data @ TU Wien)

[1] https://cocodataset.org

- 2 name nodes
- 1 administrative server h1
- 18 worker nodes (c101-c118), each with:
  - 2 XeonE5-2650v4 CPUs with 24 cores (= 48 cores)
  - 256 GB RAM
  - 4x4 TB HDD
- 10 Gbit uplink

2.2 Design Choice

In figure 1, we show key points of how our application works when it comes to remote execution. First, we pull our Docker image from Docker Hub, secondly, we load our pre-trained model from Tensorflow Hub. Then, we use a POST request to upload our images to the cluster, and finally, cluster returns our inference on all the uploaded files by a single GET request.

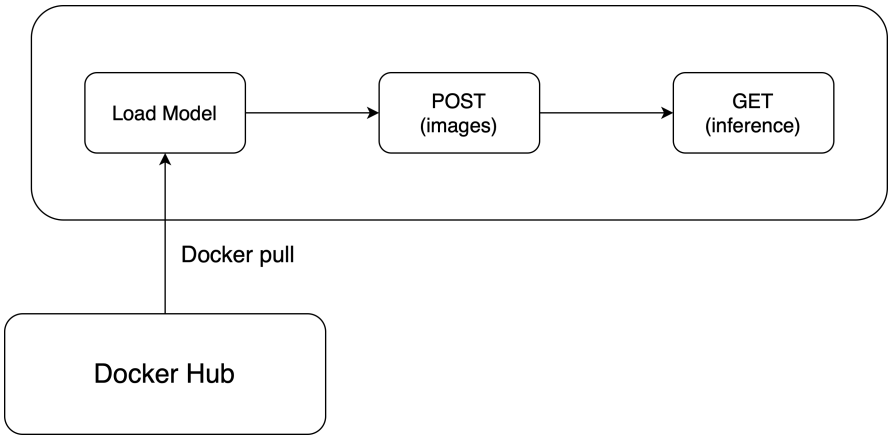

Figure 1: Design choice schema

2.3 Implementation

The object detection implementation is based on the Tensorflow object detection tutorial provided in the lecture.[2] This makes use of the pre-trained `FasterRCNN+InceptionResNet V2` detector module which is known to have reasonably high accuracy. The model is contained in the Docker image so that we do not need to download the whole model every single time it works on an image.

2.4 Commands for container deployment

Below there are the commands needed for container deployment:

```
# Those commands can be used to run the Docker container locally
# Change ports to 5025 to run it on the cluster
(in docker run change to 5025:5000)

# Fetches docker container from docker hub
docker pull fidelisus/code_offloading_object_recognition

# Runs the container
docker run -d -p 5000:5000 fidelisus/code_offloading_object_recognition
```

Using these commands, Docker image is downloaded from the Docker hub and run locally.

[2] https://www.tensorflow.org/hub/tutorials/object_detection

2.5 Interesting Facts

- We were very satisfied with the flexibility of Docker, as it was straightforward to use and worked without mayor issues.
- When deploying the application, one has to take into account that loading the model takes time (even up to 2 minutes locally). Therefore, it is important to run the application beforehand, so that the model can be loaded.

3 Results

In this section, we present experimental results that give insights into our application performance on the local machine and cluster (remote execution), results are displayed in the tables below for two different datasets. To ensure that our application performance is comparable between local machine and cluster, transfer time is included in both tables.

The second column, total time (in hours), represents the total time needed for our application to make inference on one particular dataset. The third column, inference time per file (in sec), represents the average time needed to process each image in our dataset. Average inference time per file does not include transfer time to ensure that model performance is comparable. Lastly, the transfer time (in sec), includes the total transfer time needed for the dataset to be uploaded to the cluster. Transfer time was calculated as the time to make the POST requests to upload the files.

Due to the hardware limitations, we were only able to obtain results from small and medium datasets for object detection. The model is quite big and it took us about 18 seconds to make the inference for one file in the dataset.

Execution	Total time (in hours)	Inference time per file (in sec)	Transfer time (in sec)
Local	01:27	17.58	0
Remote	00:14	02.77	16.03

Table 1: Results for Object-detection-SMALL.zip dataset

Execution	Total time (in hours)	Inference time per file (in sec)	Transfer time (in sec)
Local	10:46	18.72	0
Remote	01:40	02.83	138.62

Table 2: Results for Object-detection-MEDIUM.zip dataset

4 Discussion and Conclusion

Since we were not able to conduct our experiment on all four datasets that were provided on TUWEL due to the technical limitations, we found out that total time increases linearly. It sounds logical, as we make the inference sequentially. It means that it is enough to conduct our experiment only on one dataset and the conclusion stays the same, additionally, we can calculate an approximate total time for other datasets with different sizes.

To give the best possible answer if offloading execution on the remote cluster is worth it, we need to define three key factors:

- Model complexity
- Connection speed
- Local computing power

In our case, it is clear that offloading execution on the remote cluster is worth it, because the model we used in our application, has relatively high complexity based on the fact that it takes almost 9 times more time to obtain inference per file locally. In addition, the connection speed is good which allows for fast transfer time, and local computing power is relatively low when it comes to image processing.

To improve the performance of the local execution, the first factor, model complexity has to be relatively low, and local computing power is desired to be high. On the other side, to improve the performance of the remote execution, the connection speed is the key factor to improve.

There are scenarios where it is better to execute the application locally. For example, in the scenario where we have low model complexity, unreliable internet connection, and high local computing power, it will not be worth it to offload execution on the remote cluster because uploading files to the cluster may take more time in total than just executing the application locally. Therefore, the decision if offloading execution is worth it depends on these three factors.