Final assignment of
"Management and Analysis of Physics Datasets"

Session I

For the assignment you need to use a Linux machine. You can use a Virtual Machine (VM) from cloudveneto.it. You find the list of machines here:

https://cloud-areapd.pd.infn.it/dashboard

Remember you have to log in with your username/password on the gate first:

$ ssh <username>@gate.cloudveneto.it

Then you can access a machine as root:

$ ssh root@10.67.22.26

And then become user:

$ su <username>

PART A

Install and launch a REDIS server on the machine, as we have done in the lecture on June the 6th:

As root, download the REDIS client from https://redis.io/download (you can use version 5.0.5) into /tmp of your VM, unpack it and build it (see its README file). The directory might be already there if someone else is working on your same VM or if you have done this already in class on the same VM.

Launch the REDIS server. Remember you need to launch it in background so that you can continue working on the same shell:

$ pwd

/tmp/redis-5.0.5/src

$ ./redis-server &

[1] 1438

You can kill the server by killing the process. In this case process ID (PID) = 1438.

$ kill -9 <PID>

You can find the PID using the ps command:

```
$ ps -elf | grep redis
0 S root  1438  1405  0  80   0 - 44489 ep_pol 12:15 pts/0    00:00:00 ./redis-server *:6379
```

Now you should start working as regular user and set the environment to use the redis clients:

```
$ su <username>
$ export PATH=/tmp/redis-5.0.5/src:$PATH
```

If you did correctly you should be able to use the REDIS clients

```
$ redis-cli
127.0.0.1:6379> quit
```

PART B

Implement the cloud.sh library following all steps of Exercise 3.1.X in

https://apeters.web.cern.ch/apeters/csc2018/CloudStorage.html

but do not upload the 127 pictures as mentioned in 3.1.10. What you should do instead is to create a 1 kb file and upload it one hundred times (100) using different names.

****
HINT:
You better use a bash "for loop":

```
$ for ((i=1;i<=5;i++)); do echo $i ‘uuidgen`; done
```

Notice that the "uuidgen" command generates a random string, so you can use that to generate the name.
****

****
HINT:
Remember also the way the "cloud_upload" function works

```
$ cloud_upload /etc/passwd `uuidgen`
==> Uploading /etc/passwd with hash e7d66019a458c0b52ace76a29af6db149701ff85 to
DHT location 8
OK
```

****

****
HINT:
To create the 1kb file, remember the way the “dd” command works.
****

If you want to re-initialize your REDIS DB (in case you fill it with unexpected garbage) you can use

```
$ redis-cli -h `hostmap 1` -p `portmap 1` -n `dbmap 1` FLUSHALL
```

PART C

You now repeat part B, but you upload more files. Measure the performance (the execution time) of cloud_ls (see 3.1.14) in the following conditions:

1) With 100 items in the storage
2) With 500 items in the storage
3) With 1,000 items in the storage
4) With 5,000 items in the storage
5) With 10,000 items in the storage
6) With 100,000 items in the storage

Every time remember to re-initialize the database.

****
HINTs: the “time” command produces a very verbose output. You can apply a format to make it more friendly. For example here I time the “uuidgen” command and print its timing:

```
$ /usr/bin/time -f "Timing: %E" uuidgen
60f47e2a-b688-4110-9df3-56068eef2643
Timing: 0:00.00
```

****

Use a python notebook to plot the performance as a function of the number of items. How does it scale?

PART D

Upload one million (1,000,000) files!!! While doing this, measure the time of each file upload and plot the value as a functions of #items using a notebook.

****
HINT:

It is useful to store the output on a file, appending the results (the 2>&1 at the end of the command is magic, you need to use it) .

```
$ time -f "Timing: %E" uuidgen >> /tmp/a.a 2>&1
$ time -f "Timing: %E" uuidgen >> /tmp/a.a 2>&1
$ cat /tmp/a.a
68a2b13c-fd0f-4f40-8987-f1988064ab62
Timing: 0:00.00
30312853-7eb5-47fa-86c2-a56444f2b211
Timing: 0:00.00
```

You should know how to import a file in a pyton notebook, read line by line and extract the timing information.
****

How does the upload time changes with the # of items in the system?
Measure the time of cloud_ls and add this measurement to the plot in part C. Any change of behavior?

PART E

Answer the following questions:

1) How do you think the measurements in PART C and PART D would be affected if you use a bucket (see section 3.2)? You can try it or give a theoretical answer based on what you learned. Both are valid options
2) Imagine you implement security and for each function you call you have an authentication overhead of 1 second. How do you think your measurements will be impacted? Again, you can try this directly (you can use the unix "sleep 1" command) or give a theoretical answer based on what you learned. What would you do to limit the impact of security overhead if you were the developer of REDIS?
3) The storage we deployed using REDIS is very simple, there is no redundancy. How would you modify your service and your client to reduce the probability of data loss?

Session II

PART A

Answer the following question:
Describe in a schematic way the GRID and the Cloud layers in a distributed system commenting on the differences. (max 1 page)

PART B

Solve the following exercises by using the dataset available at the following link:
https://www.dropbox.com/s/75aqeywzxsu6hqt/dataset_sampled_handwritten.csv?dl=1

The dataset contains a huge amount of handwritten characters (a-z, English alphabet).
Each row is structured as following:

- letter column (column 0): contains the character represented by the row (ex: 'a', 'b'...)
- label column (column 1): contains integer encoding for the character represented by the row. Letter 'a' is encoded with 0, b is encoded with 1 and so on until z that is encoded with 25
- From column 2 to the end: each column represents a pixel value of a 28x28 image in grayscale format, similar to what we have seen in class for the handwritten digits recognition task

The assignment is:

1. Create your cluster with fixed number of workers, equal to 2.
2. Load the remote dataset available at the link above.
3. Count the number of examples for each alphabet character and plot them.
4. Record the time necessary to accomplish point 3, do different repartitions of data (Use 3-4 different numbers of partitions). Record the new execution times and explain their behavior.
5. Filter the data in order to get a subsampled dataset that contains only rows for character 'a' and 'z'. Print the computation graph and comment the behavior.
6. Take the subsampled dataset obtained in point 5 divide it in training (80%) and test (20%) sets.
7. Use the MinMaxScaler with IncrementalLearning, MLPClassifier and ParallelPostFit in order to training and test a simple neural network over the cluster (lecture 4/5). Take in consideration the reached accuracy and the execution time.
8. Repeat from point 1 to point 7 changing the number of workers to 4, 6, 8, 10 (**larger number of workers must be implemented with multiple machines (>= 6)**). Compare the results and the times and explain the behavior.
9. Use the distributed Tensorflow. Use the MinMaxScaler with Convolutional Neural Network that you can find in the function "*def build_deep_cnn_neural_network()"* on the Notebook "*Distributed Deep Learning on CPU-GPU*". Pay attention to the *input_shape.*

Hints:

- ***Pay attention to the shape of the data, reshape them in the correct way!***
- ***The first two columns of the data represent the character and the labels, while the columns between pixel_1 to pixel_784 represents the pixels of the images.***