✓ Terminé

Lab Session 1 - HDFS and MapReduce

Parts of this were adapted from Prof. David Auber, from the Université de Bordeaux (link)

First steps with HDFS

Have a look at the list of commands available at HDFS:

```
hdfs dfs
```

For more information on a given command, you can use:

```
hdfs dfs -usage [COMMAND]
```

There are hdfs-versions of the traditional `mkdir`, `ls`, `chmod`, etc.

- Try exploring the folders and files already present in the cluster's HDFS.
- Create a new folder for yourself under `/user/[YOUR USERNAME]/[FOLDER NAME]`
- Add a file to your folder (send it to HDFS) using the `hdfs dfs -put` command.
- Where is your file stored? In how many pieces?

```
hdfs fsck / -files -blocks -locations
```

- Obtain the `/user/fzanonboito/CISD/worldcitiespop.txt` file (from HDFS to the machine you are using) with the `hdfs dfs -get` command.

First steps with MapReduce

Run the Pi code, provided with Hadoop as an example:

```
yarn jar \ /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar \ pi 10 1000
```

Take a look at the applications that were recently submitted to yarn, try to find the one you executed.

```
 yarn application -list -appStates ALL
```

The Word Counter

Recover and run the MapReduce code `mapreduce_wordcounter.tgz` (available in our Moodle page). It implements the word count algorithm we discussed earlier. To compile it:

```
tar xzf mapreduce_wordcounter.tgz
cd mapreduce_wordcounter
mvn package
```

A `target/` folder will have been created, containing a jar you can submit to yarn. You will need to provide two parameters, the input and output paths, both in HDFS. As input, use the file already present in `/user/fzanonboito/CISD/LesMiserables.txt`, and as output a new (non-existing) folder under `/user/[YOUR USERNAME]`.

- Look at the code, found in `src/main/java/`. Identify all the information given by the programmer to the MapReduce engine.
- Inspect the newly created output.
- Look at the counters that are shown in the console at the end of the job, specially at the File System Counters. Compare them to the sizes of the input and output. What do they mean?
- Add a combiner to your code. In fact, you will not need to write a new class, you can reuse something. To add a combiner, add `job.setCombinerClass([CLASS])` to your main. If you do not remember/know what combiners are, there is a nice explanation here
- Compare the counters obtained with both versions of the code.

MapReduce in Python

In the next exercise, you can write your code in Java, which is the "traditional" way of writing MapReduce programs. If you want, you can try working in Python. To run a MapReduce code in Python, you have to write mapper and reducer codes that communicate through STDIN and STDOUT. Then we run them by using the Hadoop Streaming API, which connects everything.

In `mapreduce_wordcounter_python.tgz` , you'll find the Word Counter solution in Python, which was copied from here . To execute it, simply run

```
hadoop jar /usr/hdp/3.0.0.0-1634/hadoop-mapreduce/hadoop-streaming.jar \
-file [complete path to the mapper] \
-mapper [complete path to the mapper] \
-file [complete path to the reducer] \
-reducer [complete path to the reducer] \
-input [HDFS path to the input data] \
-output [HDFS path to the output]
```

For more options in this command, see documentation. It is also possible to pass a `-combiner` in this command.

How does the performance of the Python version compare to the one written in Java?

World city populations

Inspect the `worldcitiespop.txt` file (you obtained in the first exercise). It contains information about cities, including their population. In the first line of the file, you will find a description of all columns. Beware: some cities have an unknown population, those lines have "" for that column. Write a MapReduce program that receives that file as input (use the one that is in HDFS, in `/user/fzanonboito/CISD/worldcitiespop.txt`) and counts the number of cities by order of magnitude of their populations. The order of magnitude is to be calculated as Math.pow(10, (int) Math.log10(population)). To each order of magnitude, we want the number of cities, average, maximum and minimum population. Your output (which must be written to a file) should look like this:

```
        count   avg max min
1   5   7   8   7
10  174 55  99  10
100 2187    570 999 100
1000    20537   4498    9998    1000
10000   21550   30600   99922   10001
100000  3248    249305  997545  100023
1000000 269 2205586 9797536 1001553
10000000    10  13343569    31480498    10021437
```

To write the first line to the file, you may add a setup method to your Reducer class. Details can be found here

Write another MapReduce program who receives the same file as input. It will be useful to add the cleanup method to your Reducer and/or Mapper (see documentation here and here). The output file must contain the top K most populated cities from the input file. An useful class for this may be TreeMap (see documentation). Here is the top 10:

```
31480498    tokyo
14608512    shanghai
12692717    bombay
11627378    karachi
10928270    delhi, new delhi
10443877    manila
10381288    moscow
10323448    seoul
10021437    sao paulo
9797536 istanbul
```

- In the top-K code, what is the minimum amount of data we must send from mappers to the reducer?
- The solution for top-K can only work with a single Reducer task. Why is that?
- (Extra) modify your code from the previous exercise to receive K as an argument and output the Top K (instead of only having K = 10). To pass arguments to Mapper/Reducer, see this (we are using the new API).

Modifié le: Friday 28 October 2022, 10:17