ENSEIRB-MATMECA + Université de Bordeaux - Master 2 CISD

Supports pour le Traitement de Données

Francieli Zanon Boito

Lab Session 1[1]
HDFS and MapReduce

1. Accessing the cluster

We'll use the 10 Mistral nodes, which will have been reserved and configured by your teacher with the Hadoop environment.

```
ssh [username]@formation.plafrim.fr
ssh plafrim
```

That will take you to the mistral01 node, from where you will work. To set the environment variables required to give you access to the Hadoop cluster:

```
source /var/tmp/bigdata/user-env.sh
```

You will also need to change some settings in order to use maven to compile your programs (you just need to do it once for the whole semester). If the `~/.m2` directory does not exist at first, you can create it with `mkdir ~/.m2`

```
cp /var/tmp/settings.xml ~/.m2/settings.xml
```

2. First steps with HDFS

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

```
hdfs dfs
```

[1] Parts of this were adapted from Prof. David Auber, from the *Université de Bordeaux* https://www.labri.fr/perso/auber/BigDataGL/index.html

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 folder for yourself at `/users/[YOUR USERNAME]`.
- 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 `/data/worldcitiespop.txt` file (from HDFS to the machine you are using) with the `hdfs dfs -get` command.

3. First steps with MapReduce

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

```
yarn jar /var/tmp/bigdata/hadoop/share/hadoop/mapreduce/
hadoop-mapreduce-examples-2.8.1.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
```

4. The Word Counter

- Recover and run the MapReduce code available at `/var/tmp/mapreduce_wordcounter.tgz`. It implements the word count algorithm we discussed earlier. To compile it:

```
cp /var/tmp/mapreduce_wordcounter.tgz ~/
cd ~
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 `/data/LesMiserables.txt`, and as output a new (non-existing) folder under `/users/[`*`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[2].
- Compare the counters obtained with both versions of the code.

5. World city populations

- Inspect the `/data/worldcitiespop.txt` file. 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 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[3]:

[2] If you do not remember/know what combiners are, there is a nice explanation here: https://www.tutorialspoint.com/map_reduce/map_reduce_combiners.htm

[3] To write the first line to the file, you may add a `setup` method to your Reducer class: https://hadoop.apache.org/docs/r2.7.0/api/org/apache/hadoop/mapreduce/Reducer.html#setup(org.apache.hadoop.mapreduce.Reducer.Context)

	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

- Write another MapReduce program who receives the same file as input and an argument K[4]. Write your own class to be used as value in the output of the Mappers[5]. The output file must contain the top K most populated cities from the input file[6]. 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, can we decrease the amount of intermediate data sent from mappers to reducers? If yes, how?
- In general, the solutions to both MapReduce programs you wrote in this part can only work with a single Reducer task. Why is that?

[4] To pass arguments to Mapper/Reducer, see this (we are using the new API): http://www.thecloudavenue.com/2011/11/passing-parameters-to-mappers-and.html

[5] See the instructions in Section 3 of this link: https://javadeveloperzone.com/hadoop/hadoop-create-custom-value-writable-example/

[6] An useful class for this may be TreeMap: https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html