« Big Data »

CentraleDigitalLab@LaPlateforme - 2023

Lab-1 Part-2 : Distributed programming in Spark

Gianluca Quercini & Stéphane Vialle

We'll execute Spark programs on the distributed environment of a Spark-HDFS cluster at CentraleSupélec's Teaching Data Center. For details of implementation and access to distributed files, see Part 1 of the lab.

Exercise 1: Computing averages

Consider a collection of CSV files containing temperature measurements "Year, Month, Day, hour, minute, second, temperature"; the files are stored under folder **hdfs://sar01:9000/data/temperatures/** in HDFS

- temperatures_86400.csv: one measurement per day for the years 1980 - 2018.
- temperatures_2880.csv: one measurement every 2880 seconds for the years 1980 - 2018.
- temperatures_86.csv: one measurement every 86 seconds for the year 1980 alone.
- temperatures_10.csv: one measurement every 10 seconds for the years 1980 - 2018.

We want to generate pairs (Year, AverageTemperature) using a Map-Reduce approach in Spark.

Question 1.1 : First implementation.

- Copy file ***~cpu_vialle/DCE-Spark/template_temperatures.py*** to your home directory and create file ***avg_temperatures_slow.py***
- Complete this template by specifying the HDFS paths of the input and output file directories. The output directory must be the root of your home directory in HDFS.

```
#map-reduce computation
def avg_temperature_slow(theTextFile):
  avg = theTextFile                                          \
            .map(lambda line: line.split(","))               \
            .map(lambda term: (term[0],   [float(term[6])] ))  \
            .reduceByKey(lambda x, y: x+y)                   \
            .mapValues(lambda lv: sum(lv)/len(lv))
  return avg

#main code
input_hdfs_file_name = ……     #TO COMPLETE – see details in the real python source file
output_hdfs_file_name = ……  # TO COMPLETE – see details in the real python source file
sc = SparkContext()

input_text_file = sc.textFile(input_hdfs_file_name)
results = avg_temperature_slow(input_text_file)
results.saveAsTextFile(output_hdfs_file_name)
```

- Test your program on file *temperatures_86400.csv*, and verify the content of the output file:

```
spark-submit --master spark://sarXX:7077
             avg_temperatures_slow.py  temperatures_86400.csv
```

As a comparison, here are the key-value pairs that you should obtain for the years 2013-2018 as (year, average) :

```
(u'2013', 8.159945205479453)        (u'2016', 8.337049180327872)
(u'2014', 7.41616438356164)         (u'2017', 7.345287671232872)
(u'2015', 7.788328767123292)        (u'2018', 8.23512328767123)
```

- Complete the table below for three different input files:

(short) Input file name	*86400*	*2880*	*86*	*10*
File size (MB)	0,36	10,5	9,0	3000,0
Exec time (s)				

To get the computation time look for a line in the Spark output that looks as follows:

INFO DAGScheduler: Job 0 finished: runJob at SparkHadoopWriter.scala:78, took 3.478220 s

- How do you justify the computation time, knowing that files *temperatures_2880.csv* and *temperatures_86.csv* have a comparable size?

Question 1.2: second implementation

- Copy your file ***avg_temperatures_slow.py*** to a file ***avg_temperatures_fast.py***
- Complete the function ***avg_temperature_fast*** by modifying the three lines labeled « TO DO » below, to implement a MapReduce computation that parallelizes the computations better and decreases the network traffic between *spark-executors* :

```
def avg_temperature_fast(theTextFile):
  avg = theTextFile                                \
        .map(lambda line: line.split(","))         \
        .map(lambda term: ……)                      \          TO DO
        .reduceByKey(lambda x, y: ……)              \          TO DO
        .mapValues(lambda x: ……)                                TO DO
  return avg
```

- In the main code (at the end of the source code) don't forget to call function ***avg_temperature_fast*** (instead of ***avg_temperature_slow***).
- Complete the table below for **4** different input files:

(short) Input file name	*86400*	*2880*	*86*	*10*
File size (MB)	0,36	10,5	9,0	3000,0
Exec time (s)				

- Compare the execution times to the ones obtained in question 1.1. Where do you think your code will spend most of its time? Where did you gain in performances?

Exercise 2: Computation of averages and standard deviation on large data series

We use the same temperature files as in exercise 1, but now we want to obtain tiplets such as : *(Year, AverageTemperature, StandardType)*.

We can express the standard deviation of N values x_i (with: $0 \leq i < N$) with the following formula:

$$\sigma = \sqrt{\overline{x^2} - \bar{x}^2} = \sqrt{\frac{\sum_{i=0}^{i=N-1}(x_i{}^2)}{N} - \left(\frac{\sum_{i=0}^{i=N-1} x_i}{N}\right)^2}$$

This standard deviation formula is much more "parallelizable" than the usual formula, and we adopt it in this lab. However, it is numerically unstable! Therefore it is not used in the real parallel computations.

Question 2.1: Implementation and validation

- Propose a MapReduce approach and a Spark code that solves the problem with a fast execution time for a large data series.
- Copy again file **~cpu_vialle/DCE-Spark/template_temperatures.py** to create the Python script ***stddev_temperatures.py***
- Implement your solution, then test it on file *temperatures_86400.csv*. Verify the content of the output file.

  As a comparison, here is the key-value pairs that you should obtain for the years 2013-2018 in the format (year, (average, standard_deviation)) :

```
(u'2013', (8.159945205479453, 12.095467213807215))
(u'2014', (7.41616438356164, 12.000489970606273))
(u'2015', (7.788328767123292, 11.422338877224695))
(u'2016', (8.337049180327872, 11.508239269066483))
(u'2017', (7.345287671232872, 11.692122262422554))
(u'2018', (8.23512328767123, 11.458240836489342))
```

Question 2.2: performance measurements

- Execute your program to complete the table below.

Input file name	*86400*	*2880*	*86*	*10*
File size (MB)	0,36	10,5	9,0	3000,0
Exec time (s)				

- Compare the execution times with the ones obtained in exercise 1. What do you conclude?