« Big Data » course - CentraleDigitalLab@LaPlateforme – 2023-2024

Tutorial 1: Basic Map-Reduce algorithms in Spark

Stéphane Vialle & Gianluca Quercini

Exercise 1: Calculating averages on data series

Question 1.1: processing of small data sets

Consider a CSV file containing "*Year*, *Month*, *Temperature*" measurements, with only one temperature measured per month (so theoretically 12 measurements per year...).

We would like to generate pairs (Year, TemperatureAverage) using a Map-Reduce approach in Spark.

- Describe a Map-Reduce approach that solves the problem, without enriching the data during the process (manipulate only years and temperatures).
- Propose a Spark implementation by completing the following code:

```
#map-reduce computation ------------------------------------------
 def avg_temperature(theTextFile):
     avg = theTextFile \
             .map(lambda line: line.split(",")) \
             . ……..            # nb of line is not limited – TO DO
     return avg

 #main code -------------------------------------------------------
 Input_hdfs_file_name = ……    #details in the real python source file
 Output_hdfs_file_name = ……   #details in the real python source file

 sc = SparkContext()

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

- Does your Spark code maintain RDDs until the last step?
- Does your Spark code contain Narrow or Wide operations?
- What are the shortcomings of this algorithm?
- We now assume that there is only one key (all temperatures were collected in the same year).
    - Let Ns be the number of Spark executors, Nt the total number of temperatures read. What will happen if the amount of data increases?
    - Estimate the volume of data injected into the network during the shuffle.

Question 1.2: processing large data series

We now consider a large CSV file (stored taking full advantage of a distributed HDFS file system), containing "*Year*, *Month*, *Day*, *Hour*, *Minute*, *Second*, *Temperature*" measurements. In some years, there may be as many as one measurement per second.

As before, we want to generate pairs (*Year*, *TemperatureAverage*) using a Map-Reduce approach in Spark.

- How many measurements can there be in one year at most?
  Can we continue with the algorithmic approach of question 1?
- Propose an approach better suited to these large datasets, and a Spark implementation.
- Does your Spark code maintain RDDs? Does it favor Narrow operations? Which operations modify data partitioning?

Exercise 2: Calculating averages and standard deviations on large data series

We start again from question 2 of exercise 1, and our "*Year*, *Month*, *Day*, *Minute*, *Second*, *Temperature*" measurements in a large CSV file.

But now we want to obtain triplets of values: (*Year*, *MeanTemperature*, *StandardDeviation*).

We can express the standard deviation of N values x_i (with: $0 \leq i < N$) according to two equivalent definitions:

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

Question 2.1: choosing a mathematical expression

Which expression of the standard deviation do you find most exploitable in a Map-Reduce approach and a Spark implementation (and why)?

Question 2.2: Spark algorithm and implementation

Propose a Map-Reduce approach and Spark code to solve the problem for large datasets.