POLITECNICO
MILANO 1863

Middleware technologies for distributed systems
Project 5

2020-2021

Authors:
Andrea Falanti
Federico Ferri
Abanoub Faltaous

Professors:
Prof. Alessandro Margara
Prof. Luca Mottola

Contents

1 Introduction

The project requires to implement a system to study the evolution of the COVID-19 cases worldwide. In particular, it is required to compute:

- Seven days moving average of new reported cases, for each county and for each day
- Percentage increase (with respect to the day before) of the seven days moving average, for each country and for each day
- Top 10 countries with the highest percentage increase of the seven days moving average, for each day

The dataset used is the csv format provided in this link: https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide.

The records contained in the csv are organized in weeks and contains multiple fields, which some of them are not relevant for the scope of the problem.

2 Implementation

To implement this project we decided to use Apache Spark, because it's indicated to operate on big amount of data in a reliable and fast way. Spark provides functionalities to schedule automatically custom stages of operations, in a similar way to map/reduce systems, so that the developer must only define the operations to perform, while the parallelization and fault tolerance aspects are handled by the Spark engine. Apache Spark is imported into a Java project, by defining the imports in the Maven file.

The program accepts four arguments from command line:

- address of the Spark master instance
- path of the dataset
- boolean flag (”true” or ”false”) to indicate if output should be print on console or file (csv)
- save path, relevant only if output is saved on file

All operation are implemented by using the Spark SQL module, that allows to optimize the operations using techniques similar to the ones used in DBMS. The first step is generating a dataframe from the records read from the csv data file. A preprocessing step is also required to separate the records (organized in the csv as weeks) into days, generating therefore 7 rows from each one of the csv. Performing this operation is a bit tricky in Spark, because dataframes are meant to be immutable, therefore the solution we used is to add columns for date and cases and then explode the row into multiple rows with the 'explode' Spark function. After reading and preprocessing the data, we select only the columns that are required for our purposes, that are:

- date
- daily_cases
- countriesAndTerritories

To compute the required indexes, we need to perform some operations that require to iterate on multiple rows of the dataframe. A nice way to perform this kind of operations is to use the Window class.

To perform the first operation and second operation we created a WindowSpec partitioned by *countriesAndTerritories* field and ordered by *date*. To compute the moving average for each day, it's necessary to consider the *daily cases* of the row itself and the ones of previous six rows, then average them. If there are not six records before the considered row, the average is performed anyway but only on the present rows. To compute the variation percentage, we firstly compute the variation in the *moving average* between the considered row and the previous one, then compute the percentage with the formula:

$$variationPercentage_t = \frac{variation_t}{mov\_average_{t-1}} * 100$$

To take the previous row, we just use the "lag" function on the window, with lag parameter = 1. In case there is no previous row, the variation is set to 0. Instead, if the moving average at the previous step is 0, the variation is divided with "Double.MIN_VALUE", making the variation percentage to infinity. We considered to make it to NaN instead of infinity in the case explicated before, but it would not be detected in the top 10 countries computed in the next operation if using NaN, but it could be an important event of interest like the start of an infection in a country.

For the last operation, we create a WindowSpec partitioned by date and ordered by the variation percentage computed before, ordered in a descent manner. We can then select the first 10 rows of each partition of the window (Spark allows to get the row index of a partition by using the 'row_number' method) and so complete the request of finding the top 10 countries with highest percentage increase per day.

3 Performance analysis

We tested the application in a distributed cluster composed by two different computers, one running the master and a worker, the other running only a worker, obviously connected to the master. During the tests, the dataset csv file has been placed in a shared directory, so that was accessible by both workers.

To improve performances, the dataframe computed in the first operation is cached, so that computations are not repeated when the second operation is performed, the same is done for the second dataframe for the third operation. When caches are not needed, they are freed to regain memory.

Time results:

- single worker: 22s
- two workers (different devices): 25s

From these results it seems that the overhead of the communication and coordination of the workers are more time consuming than the performance gains given by the parallelization. This behavior could be caused by the fact that the actual dataset is pretty small and also that we have only two workers, which is an unusual case for a distributed cluster of workers. With a larger amount of data and more workers in the cluster, we expect to have great benefit from the parallelization of the stages performed by Spark.