Analysis of COVID-19 Data

Davide Remondina, Marco Rodino, Andrea Zanin

In this project we have implemented a program that analyzes open datasets to study the evolution of the COVID-19 situation worldwide.

General structure

We used Kafka to store the Covid data and Spark to analyze it in a streaming fashion. The combination of those two tools makes our implementation very scalable: the system can be deployed on several machines to handle very large streams of real-time data.

The code is divided into 3 source files which can be compiled and deployed separately:

- `BasicProducer.java`: a Kafka Producer that publishes Covid data for a new day every 5 seconds on a topic which is used as data source for all the analysis
- `Covid.java`: a program that computes 7-day moving averages and daily percentage increase of those averages for each day and for each country
- `TopTen.java`: a program that find the 10 countries with the highest percentage increase of the 7-day moving average for each day

Covid.java

Input processing

We decided to read the necessary information from the Kafka Topic using the *Structured Streaming integration for Kafka 0.10* provided by the Spark library. In particular we used a Structured Streaming approach assuming that 1 day lasts 5 seconds to simplify testing, but we can change the `windowDuration` and `windowSlide` parameters to match a real world situation.

From line 44 to line 54 we create the schema of the database that we will read from Kafka. The fields are

- `dayCount` - the number of day passed from 3 January 2020
- `dateReported` - the day the measure are taken
- `countryCode` - the code of the country
- `countryName` - the name of the country the measure belongs to
- `countryArea` - the area of the country
- `newCases` - the new cases of covid-19 registered that day
- `cumulativeCases` - the total cases of covid-19 registered from 3 January 2020
- `newDeaths` - the new deaths caused by covid-19 during that day
- `cumuluativeDeaths` - the total death caused by covid-19 from 3 January 2020

From row 56 to row 62 we start the stream reading from the Kafka topic, from row 67 to row 71 we convert the value of the Kafka record read from json to the schema we have prepared before.

In row 74 we add the watermark, which is necessary to handle late data. In an application with real time data we would use 14 days as delay threshold; so that if some data is received with a delay of less than 14 days, the program will recompute all the necessary windows. Spark Structured Streaming can maintain the intermediate state for partial aggregates for a long period of time such that late data can update aggregates of old windows correctly.

Data analysis

From row 75 to row 90 is where most of the query is done. Our first approach was using a sliding window with duration of 7 days and slide of 1 day to compute the 7-days moving average. But in this way computing the daily increase would have been impossible using structured streaming: in order to compute the percentage increase we need the computed mean of the previous sliding window, which can be obtained from spark with the use of the function `lag(delay).over(WindowSpec)` which however is not available when using structured streaming.
Another solution solution could have been using Batch processing but this would have been much worse in terms of latency. Instead we chose to extend the window to 8 days, and for each window compute the old and the present average in order to compute the percentage increase.

Once we have computed the 7-days moving average and the percentage increase, we select only the complete windows (because we are not interested in computing the moving average of incomplete future days) and we discard the column that we are not interested to show.

From row 92 to row 98 we start writing the stream. We write the stream on console for demonstration purposes but we could easily change the code to write the resulting query on a file, on another kafka topic or other outputs.

TopTen.java

This file is used to find the 10 countries with the highest percentage increase of the seven days moving average, for each day. We reuse most of the code of the previous file, but we opted for a batch-processing instead of a streaming, since the function `over(WindowSpec)` that simplifies a lot the code is not available when using streaming. So every time we run the class TopTen we compute for each day the 10 countries with the highest percentage increase of the seven days moving average.

Until line 90 the code is identical to the previous exercises, then for each day we rank each row based on the column *Percentage* (It's the line 95, which use the windows specification written on line 92) and we keep only the row with a rank lower than or equal to 10.