POLITECNICO
MILANO 1863

Networked software for distributed systems

Analysis of COVID-19 Data

Authors:
Stefano Carraro
Stefano Fossati
Andrea Restelli

Professors:
Prof. Luca Mottola
Prof. Alessandro Margara

2022-2023

Contents

1 Introduction

1.1 Request

1.1.1 Description

In this project, you have to implement a program that analyzes open datasets to study the evolution of the COVID-19 situation worldwide. The program starts from the dataset of new reported cases for each country daily and computes the following queries:

- 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

You can either use real open datasets or synthetic data generated with the simulator developed for Project #4.

A performance analysis of the proposed solution is appreciated (but not mandatory). In particular, we are interested in studies that evaluate (1) how the execution time changes when increasing the size of the dataset and/or number of countries; (2) how the execution time decreases when adding more processing cores/hosts.

1.1.2 Assumption and Guidelines

- When using a real dataset, for countries that provide weekly reports, you can assume that the weekly increment is evenly spread across the day of the week.

1.1.3 Technologies

Apache Spark or MPI

1.2 Additional Assumptions

- Since the seven days moving average is calculated by summing up the values of the past seven days and dividing by seven, it is necessary to have a complete set of data for accurate calculation. Therefore, for the initial six days, the average is based on the available records, and as each subsequent day's data becomes available, it is incorporated into the calculation, resulting in a more accurate representation of the trend.

2 Project analysis and considerations

The objective of this project is to analyze a static dataset by executing three queries sequentially while leveraging the computational power of multiple cores/hosts for enhanced performance. In such a scenario, Spark emerges as an ideal solution, with its Spark SQL API offering a comprehensive range of functionalities to effectively address the project requirements. Notable advantages provided by the Spark SQL API include:

- seamless handling of CSV datasets through dataframes
- built-in windowing functions that facilitate data aggregation over specified time periods
- efficient and transparent distribution of computational tasks across multiple worker nodes

These features make Spark SQL an excellent framework for tackling the project requirements.

3 Design and Implementation

3.1 Input reading and preprocessing

The program reads the input dataset from csv files with two possible structures. The first type is that of a real dataset by European Center for Disease Prevention(ECDC), available here. The file contains records of daily cases of COVID-19 for each country in the world from 31/12/2019 to 14/12/2020. Here below is the header of the csv file (last columns have been abbreviated for readability).

dateRep	day	month	year	cases	deaths	country	geoId	cCode	pop2019	cExp	cum

The second type of supported dataset is that produced by the simulator developed in Project #4. Being simulated data, dates are here simply days starting from day 1, while countries are represented by integer country ids. Here below is the header of this second type of csv file:

day	country_id	non_infected	infected	immune

Both the types of csv files are preprocessed and reduced to a common structure of the dataset, where only interesting columns are maintained. To do so we exploited two different implementations of the same abstract class `AbstractPreprocessor`, `EcdcPreprocessor` for the real dataset and `SyntheticDataPreprocessor` for synthetic data generated by the simulator.

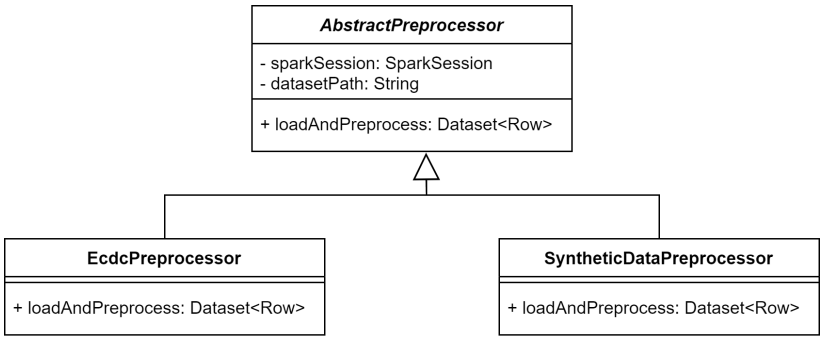


Figure 1: The two input preprocessors.

The resulting dataset has the following common structure, that is then used for the three queries.

date	cases	country
...	...	...

3.2 Data analysis

The project requires to compute three queries on the dataset.

3.2.1 Seven days moving average

Grouping the dataset by country, we exploited the built-in Spark SQL *Window* functions to consider, for each day, the previous seven days and collect the sum of the new cases over this period. With a slide of 1 day, then we compute:

$$\text{MA} = \frac{\sum_{i=1}^{7} new\_cases\_i}{7}$$

3.2.2 Percentage increase of the MA

To compute this query we exploited again the *Window* functions considering, for each day, the seven days MA of the day before:

$$\text{Percentage Increase} = \left(\frac{\text{MA}_{\text{current day}} - \text{MA}_{\text{previous day}}}{\text{MA}_{\text{previous day}}} \right) \times 100$$

3.2.3 Top 10 countries with the highest percentage increase

Finally, to compute, for each day, the top 10 countries with the highest percentage increase, we ordered the countries by decreasing percentage increase and took the first 10 records. In this way, even in case of ties, we are sure to keep 10 and only 10 countries for each day.

3.3 Caching

Since each of the last two queries were based on the results of the previous one (percentage increase of the moving average and top 10 countries with the highest percentage increase), the caching capabilities offered by Spark perfectly fitted our needs. For this reason, after each query, the results are cached to be used in the following one without recomputing the whole results from scratch. Furthermore, after the computation of the second query, the results of the first are unpersisted, and the same is done for the results of the second query after those of the third query have been computed.

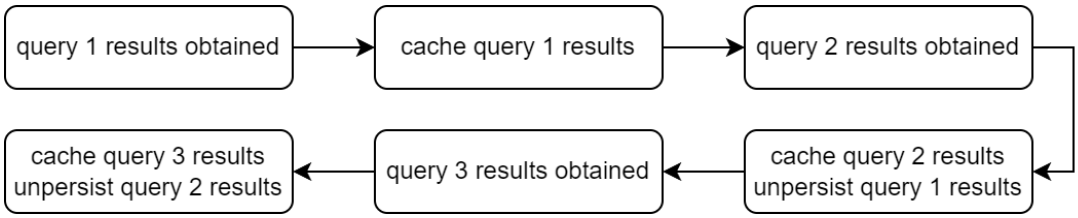


Figure 2: A diagram of the caching strategy

3.4 Writing to file

To write the results of the computed queries on file, we had to collect the DataFrame to a single partition by using *coalesce(1)*, and then write it to a CSV file. Without doing so, each partition would have written its own file, resulting in multiple files to be manually merged.

4 Performance analysis

In this section we analyze how the *execution time* varies with respect to:

- the size of the dataset and/or number of countries
- the number of processing cores/hosts

4.1 Changing the size of the dataset

To change the size of the dataset we have two possibilities:

- change the number of days records are available
- change the number of countries records are available

We applied these transformations both to the ECDC real dataset and to the synthetic dataset generated by the simulator, obtaining datasets with 50, 100, 150, ..., 350 days, and 50, 100, 150, 200 countries. The experiments were run using a single worker on a single physical machine, monitoring the execution time from the Spark history server. The results presented in Tables 1 and 2 highlight the relationship between the size of the dataset and the corresponding execution time. It can be observed that as the dataset size increases, the execution time also increases, exhibiting an almost linear relationship. This suggests that larger datasets require more time for processing and analysis.

Table 1: Varying the number of days

Day	Execution Time
50	24s
100	31s
150	32s
200	35s
250	36s
300	37s
350	45s

Table 2: Varying the number of countries

Countries	Execution time
50	31s
100	34s
150	38s
200	39s

Figure 3: Varying the number of dates

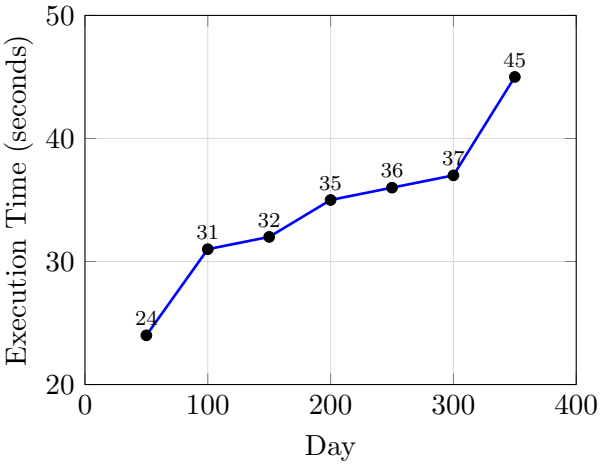
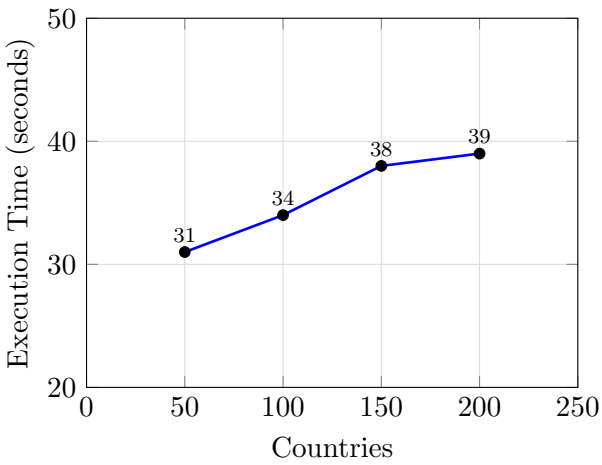


Figure 4: Varying the number of countries



4.2 Changing the number of workers

To carry out this second analysis we considered the full ECDC dataset, increasing the number of worker hosts from 1 to 3 and monitoring the execution time from the Spark history server. The results are presented in Figure 5.

As expected, we observed a decrease in the execution time as the number of available workers increased. However, the performance improvement was not as significant as initially expected, potentially due to the dataset size, which may not be large enough to fully utilize the additional computational power offered by the extra workers. To verify this claim we performed the computation with an augmented dataset obtained by the simulator of Project #3, but the results were similar. Another factor affecting the performance boost is the increase of latency caused by the introduction of more workers in the computation.

Figure 5: Varying the number of workers