Spark Covid Analysis

Bruno Morelli Cristian Lo Muto Vincenzo Martelli

Contents

0.1 Introduction

This document is the documentation of the Spark Covid Data Analysis application, which purpose is to provide a comprehensive overview of the project. Spark Covid Data Analysis is a Java-based Application that uses the Spark functionalities to analyse and process historical data about the daily number of new COVID-19 case from March 2020 to 14 December 2020 collected by European Centre for Disease Prevention and Control (ECDC). Starting from the original collection, Spark Covid Data Analysis calculates:

- 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.

0.2 Installation

The follow steps are those needed to install and run the project, both in a local and in a distributed environment (Unix machine). For the Machine that host the Master process (the process involved in managing the distribution of tasks and resources across the Spark Cluster)

1. If not already installed, install the jdk11 using the command:
   sudo apt-get install openjdk-11-jdk
2. Download the latest version of Apache Spark.
3. Extract the .tgz file.
4. Go in the directory just extracted and execute the following:

   - export SPARK_MASTER_HOST= “address-of-this-machine”
   - /sbin/start-master.sh
   - ./sbin/start-worker spark://”address-of-this-machine”:7077 (this will start a process also in the machine that host the Master)

   To run the program in a local environment put 127.0.0.1 as address of the machine.
5. To start the program run command:

   - ./bin/spark-submit “path-to-the-jar-of-the-program” spark://”address-of-this-machine”:7077 “path-to-the-data-to-be-analysed” (note that the data must have the same format of the one collected by ECDC)

   In a distributed environment is required to start this command only once all the workers are online (see next step).

For the machines that host the Worker process:

1. Download the latest version of Apache Spark.
2. Extract the .tgz file.
3. Go in the directory just extracted and execute the following:
    - export SPARK_MASTER_HOST= “address-of-master-machine”

0.3 Approach and assumption

Since the data provided from ECDC are structured their analysis is performed using Spark SQL, this brings two big advantages:

1. Writing the query is easier.
2. The using of pre-defined operator offers more opportunities for the engine to optimize the computation.

0.4 Implementation

0.4.1 Setup

The collection of data is loaded in the application using DataFrame(Dataset<Row>):

```
final List<StructField> mSchemaFields = new ArrayList<>();
mSchemaFields.add(DataTypes.createStructField( name: "dateRep", DataTypes.DateType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "day", DataTypes.IntegerType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "month", DataTypes.IntegerType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "year", DataTypes.IntegerType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "cases", DataTypes.IntegerType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "deaths", DataTypes.IntegerType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "countriesAndTerritories", DataTypes.StringType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "geoId", DataTypes.StringType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "countryTerritoryCode", DataTypes.StringType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "popData2019", DataTypes.IntegerType,  nullable: true));
mSchemaFields.add(DataTypes.createStructField( name: "continentExp", DataTypes.StringType,  nullable: true));
final StructType mSchema = DataTypes.createStructType(mSchemaFields);


final Dataset<Row> covidData = spark
        .read()
        .option("header", "true")
        .option("delimiter", ",")
        .option("dateFormat", "dd/MM/yyyy")
        .schema(mSchema)
        .csv(filePath);


covidData.drop(col( colName: "year"));
covidData.drop(col( colName: "month"));
covidData.drop(col( colName: "deaths"));
covidData.drop(col( colName: "geoId"));
covidData.drop(col( colName: "countryTerritoryCode"));
covidData.drop(col( colName: "popData2019"));
covidData.drop(col( colName: "continentExp"));
```

In this snipped of code all columns present in the original collection are imported in the DataFrame, but some of them are useless and will be dropped due to performance reasons.

0.4.2 Seven days moving average of new reported cases, for each county and for each day.

The seven days moving average of new reported cases, for each country and for each day, is calculated reading the data with a Window that partitions all the data by country and then sort those data by date. The window will cover the entries of the last 7 days of the collection at time and will move forward day by day. In this way the average of the reported case for the days in the window is calculated.

```
// Q1.  Seven days moving average of new reported cases, for each country and for each da
WindowSpec window=Window.partitionBy( colName: "countriesAndTerritories").orderBy( colName: "dateRep").rowsBetween(-6,0);
Dataset<Row> query1= covidData.select(col( colName: "dateRep"),col( colName: "countriesAndTerritories"), avg( columnName: "cases").over(window)
              .cast(new DecimalType( precision: 20, scale: 4)).as( alias: "AVG"))
       .orderBy(desc( columnName: "dateRep"));
```

0.4.3 Percentage increase (with respect to the day before) of the seven days moving average, for each country and for each day

The percentage increase of the seven days moving average is calculated starting from the previous query. First of all, the column "dayBeforeAvg" is added to the result of the first query, in this column there is the seven days moving average of the day before for each day and for each country(as before a window is used to read data of a range of day, but this time the window cover only the selected days and the one right before). Then another column is added which contain the percentage increase of the seven days moving average.

```
//Q2. Percentage increase (with respect to the day before) of the seven days moving average, for each country
//and for each day

WindowSpec window2=Window.partitionBy( colName: "countriesAndTerritories").orderBy( colName: "dateRep");

Dataset<Row> query2=query1.withColumn( colName: "DayBeforeAVG",lag(col( colName: "AVG"), offset: 1).
       over(window2));
query2= query2.withColumn( colName: "percentageIncreased",when(col( colName: "dayBeforeAVG").equalTo( other: 0),  value: "0")
       .otherwise(col( colName: "AVG").minus(col( colName: "DayBeforeAVG"))
              .divide(col( colName: "DayBeforeAVG"))
              .cast(new DecimalType( precision: 20, scale: 4))))
       .orderBy(desc( columnName: "dateRep"),desc( columnName: "percentageIncreased"));
```

0.4.4 Top 10 countries with the highest percentage increase of the seven days moving average, for each day

This query is also calculated starting from the previous one. The data of the second query are partitioned by day and sorted using the "percentageIncreased" attribute. With the function rank() to each day is assigned a value based on its rank. Then only the country with rank less than 11 are chosen to be part of the result.

```
//Q3. Top 10 countries with the highest percentage increase of the seven days moving average, for each day
WindowSpec window3=Window.partitionBy( colName: "dateRep").orderBy(desc( columnName: "percentageIncreased"));
Dataset<Row> query3=query2.select(col( colName: "dateRep"),col( colName: "countriesAndTerritories"),col( colName: "percentageIncreased"),
        rank().over(window3)
                .as( alias: "rank")).select(col( colName: "dateRep"),col( colName: "countriesAndTerritories"),col( colName: "percentageIncreased"),col( colName: "rank"))
                .where(col( colName: "rank").lt( other: 11)).orderBy(desc( columnName: "dateRep"),col( colName: "rank"));
```

0.5 Testing

The testing phase primarily focused on evaluating the performance of the application in a local environment. Three resource configuration were considered (1,4 and 8 core) and three datasets, the official one released by ECDC, one that is one-hundred bigger and one one-thousand bigger.

0.5.1 Results with the official dataset

The following images represent the result of the application with the official dataset released by ECDC, using different configuration of the hardware.

1 core

3.4.0	local-1688171470113	Covid-19	2023-07-01 02:31:09	2023-07-01 02:31:19	10 s	vincenzo	2023-07-01 02:31:19

4 core

3.4.0	local-1688171549570	Covid-19	2023-07-01 02:32:28	2023-07-01 02:32:38	10 s	vincenzo	2023-07-01 02:32:38

8 core

3.4.0	local-1688171600782	Covid-19	2023-07-01 02:33:19	2023-07-01 02:33:29	10 s	vincenzo	2023-07-01 02:33:29

With a small dataset the application work with the same performance in all configuration.

0.5.2 Results with one-hundred times bigger dataset

With a bigger dataset the situation changes, and an improvement can be seen when the number of cores increases, especially passing from 1 to 4 cores.

1 core

3.4.0	local-1688171664828	Covid-19	2023-07-01 02:34:24	2023-07-01 02:35:58	1.6 min	vincenzo	2023-07-01 02:35:58

4 core

3.4.0	local-1688171796034	Covid-19	2023-07-01 02:36:35	2023-07-01 02:37:34	60 s	vincenzo	2023-07-01 02:37:34

8 core

3.4.0	local-1688171900740	Covid-19	2023-07-01 02:38:19	2023-07-01 02:39:19	59 s	vincenzo	2023-07-01 02:39:19

0.5.3 Results with onethousand bigger dataset

When using this dataset the performance do not change a lot with the previous case, as before a big improvement can be seen when passing from 1 to 4 cores.

1 core

3.4.0	local-1688172178766	Covid-19	2023-07-01 02:42:58	2023-07-01 02:44:16	1.3 min	vincenzo	2023-07-01 02:44:16

4 core

3.4.0	local-1688172291226	Covid-19	2023-07-01 02:44:50	2023-07-01 02:45:40	50 s	vincenzo	2023-07-01 02:45:40

8 core

3.4.0	local-1688172412829	Covid-19	2023-07-01 02:46:51	2023-07-01 02:47:42	50 s	vincenzo	2023-07-01 02:47:42