Group exercise Spark y.y. 2023/2024

- Valerio Morelli
- Federica Paganica
- Federico Staffolani
- Enrico Maria Sardellini
- Simone Di Battista
- Patrice Kamdem Defo

The goal

The goal
What is the aim of the assignment?

Setting-up
How was the dataset read?

The code
How were the three queries implemented?

The goal

Dataset: CSV files are provided for each year and weather station. The number of stations varies each year, and **the files may have different fields**: the program must be able to handle this situation.

Assignments:

1. Print the **number of measurements taken each year for each station**, *sorted by year and station*.
2. Print the **10 most frequent temperatures and their relative counts** in the highlighted area, *sorted by occurrence and temperature*.
3. Print the name of the **station with the most frequent wind speed** in knots along with the corresponding count, *sorted by count, speed, and station*.

Setting-up

The goal

What is the aim of the assignment?

Setting-up

How was the dataset read?

The code

How were the three queries implemented?

The problem of reading multiple CSV files

The HDFS file system is designed to work with large files. It is therefore not surprising that Spark **is slow when reading multiple CSV files**.

However, as the documentation explains, Spark provides **several utilities to speed up** this process:

- The ***Load*** method of the *DataFrameReader* class also **accepts a list of file paths** instead of just one.
- The ***recursiveFileLookup*** option enables deep directory scanning.
- **Glob patterns** can be used to specify which files to include when reading the contents of a directory (e.g. *"hdfs://Dataset/*.csv"* identifies the list of file paths within the *Dataset* directory).

The problem of mismatched columns in CSVs

Unfortunately, while the previous methods are very efficient, they can only be used **if the CSV schema is fixed**, and, as the challenge warns, this is not a given.

As I explained in **this** Stack Overflow question, this is what happens when using the previous approach:

file1.csv

A	B	C
1	2	5
3	4	6

file2.csv

A	C
7	8

load(".csv")*

A	B	C
1	2	5
3	4	6
7	**8**	**NULL**

expected

A	B	C
1	2	5
3	4	6
7	**NULL**	**8**

The solutions we have developed

1. **Ignore the problem** – Since **the problem only occurs if one of the columns of interest** (*LATITUDE*, *LONGITUDE*, *WND* and *TMP*) **changes position** in at least one of the CSV files, and since this does not seem to be the case in the sample dataset provided, where instead the changing columns seem to be in the last indices, one might think of ignoring such a problem and reading the whole dataset like this: ***df = load(f"{DATASET_PATH}/*/*.csv")***. This is **the most efficient solution**.
2. **Read each file individually** – To ensure that the code works independently of the column positions, one could think of reading each CSV file individually and then merging them using the [*unionByName*]() method, with the ***allowMissingColumns*** parameter set to *true* to automatically fill the missing columns with *null* values. However, **albeit being an effective solution, it is the less efficient one**, due to the multiple read operations.
3. **[Read groups of CSV files]()** – Finally, the solution we have chosen is a mixture of the first two. Instead of reading each file individually, **we first analyze their headers**, then **group them by compatibility**, and **finally merge them**.

Benchmark results

The third solution **maintains the correctness** of the second but **performs much better** based on the number of groups identified.

The benchmark produced the following average results after several tests on a low-end laptop:

Dataset	*37 MiB, 38 files*	*2.4 GiB, 2432 files*
Solution 1	6.7s	260s
Solution 2	20.1s	>1307s
Solution 3	12.4s	607s

Extract station name and year fields

From the location of the single CSV file, the **station name** and **year of measurement** must be extracted as follows:

hdfs://<dataset_path>/<year>/<station_name>.csv

The hidden [*_metadata*]() column proved to be very useful for this. In particular, the *year* and *station* columns were created using the *file_path* and *file_name* fields as follows:

```
def read_csv(csv_files=None):
    return spark.read.format('csv') \
        .option('header', 'true') \
        .load(csv_files) \
        .withColumn('year', split(col('_metadata.file_path'), '/')) \
        .withColumn('year', col('year')[size('year') - 2]) \
        .withColumn('station', split(col('_metadata.file_name'), '.csv')[0])
```

The solution

The goal
What is the aim of the assignment?

Setting-up
How was the dataset read?

The code
How were the three queries implemented?

First Query

```
def op1(df):
    """
    Op1: print out the number of measurements taken per year
    for each station (sorted by year and station)
    """
    result_df = df \
        .select(['year', 'station']) \
        .groupBy('year', 'station') \
        .agg(count('*').alias('num_measures')) \
        .orderBy('year', 'station')
    write_to_file('op1', result_df)
```

QUERY

Projection on the columns of interest to reduce the data frame size

Counting the rows and ordering them as requested

EXPORT RESULTS

Output 1

```
+----+-----------+------------+
|year|    station|num_measures|
+----+-----------+------------+
|2000|99999994988|         316|
|2000|99999994989|         316|
|2000|99999994991|         316|
|2000|99999994992|         316|
|2000|99999994993|         316|
|2000|99999994994|         316|
|2001|99999994988|         375|
|2001|99999994989|         375|
+----+-----------+------------+
only showing top 8 rows

Op 1 terminated in 2.2416326999664307 s.
```

Second query

```python
def op2(df):
    """
    Op2: print the top 10 temperatures (TMP) with the highest number of occurrences and count recorded
    in the highlighted area (sorted by number of occurrences and temperature)
    """
    result_df = df \
        .withColumn("LATITUDE", col('LATITUDE').cast(FloatType())) \
        .withColumn("LONGITUDE", col('LONGITUDE').cast(FloatType())) \
        .withColumn("TMP", split(col('TMP'), ',')[0].cast(FloatType()) / 10) \
        .select(['LATITUDE', 'LONGITUDE', 'TMP']) \
        .filter((col('LATITUDE') >= 30) & (col('LATITUDE') <= 60) &
                (col('LONGITUDE') >= -135) & (col('LONGITUDE') <= -90)) \
        .groupBy('TMP') \
        .agg(count('*').alias('num_occurrences')) \
        .orderBy(col("num_occurrences").desc(), col("TMP").asc()) \
        .withColumn('Location', lit('[(60,-135);(30,-90)]')) \
        .select(['Location', 'TMP', 'num_occurrences']) \
        .limit(10)
    write_to_file('op2', result_df)
```

QUERY

EXPORT RESULTS

Temperature in human readable format

Projection on the columns of interest to reduce the data frame size

Counting the rows and ordering them as requested

Output 2

```
+--------------------+-----+---------------+
|            Location|  TMP|num_occurrences|
+--------------------+-----+---------------+
|[(60,-135);(30,-90)]|999.9|           4468|
|[(60,-135);(30,-90)]| 14.1|            455|
|[(60,-135);(30,-90)]| 14.4|            409|
|[(60,-135);(30,-90)]| 14.9|            372|
|[(60,-135);(30,-90)]| 14.5|            360|
|[(60,-135);(30,-90)]| 13.6|            358|
|[(60,-135);(30,-90)]| 16.0|            346|
|[(60,-135);(30,-90)]| 15.6|            344|
|[(60,-135);(30,-90)]| 15.7|            335|
|[(60,-135);(30,-90)]| 13.7|            329|
+--------------------+-----+---------------+
only showing top 10 rows

Op 2 terminated in 2.1042284965515137 s.
```

Third query

```
def op3(df):
    """
    Op3: print out the station with the speed in knots and its count
    (sorted by count, speed and station)
    """
    result_df = df \
        .select(['station', 'WND']) \
        .withColumn('WND', split(col('WND'), ',')[1]) \
        .groupBy('station', 'WND') \
        .agg(count('*').alias('num_occurrences')) \
        .orderBy(col("num_occurrences").desc(),
                 col("WND").asc(), col("station").asc()) \
        .limit(1)
    write_to_file('op3', result_df)
```

Projection on the columns of interest to reduce the data frame size

In the WDN column, only the wind speed in knots is taken

Counting the rows and ordering them as requested

QUERY

EXPORT RESULTS

Output 3

```
+-----------+----------+----------------+
|    station|WDN_knots|num_occurrences|
+-----------+----------+----------------+
|72410499999|         9|           29108|
+-----------+----------+----------------+
only showing top 1 row

Op 3 terminated in in 1.1821048259735107 s
```