Project 2: Advanced Infrastructures for Data Science

Jan Janiszewski, UCNumber:2023198681
Gabriel Diaz, UCNumber:2017278436
University of Coimbra
Coimbra, Coimbra, Portugal

1 INTRODUCTION

All the assignments described in the following section are related to Big Data processing tools such as Hadoop and Spark.

1.1 Data processing tasks

Assignments largely involve performing the same data processing with different tools. To avoid repetition, the data processing tasks are described here:

(1) Average Number of Friends by Age
(2) Minimum Temperature Per Capital
(3) Sort the Word Frequency in a Book
(4) Sort the Total Amount Spent by Customer
(5) Least\Most Popular Superhero

In some cases output of tasks was truncated to not extend maximal report length.

2 ASSIGNMENT 1

2.1 Goals of assignment

Goals of this assignments were to solve tasks #1, #2, #3 and #4 using Hadoop MapReduce.

2.2 Solution #1

Code for this task can be found in task1.py file. The code consists of a single MapReduce step, where the mapper extracts age and number of friends, and the reducer computes the average number of friends for each unique age.

Running a file (with path to *fakefriends.csv* as an argument) produces following output:

```
"54"   278.0769230769231
"55"    295.53846153846155
"56"    306.6666666666667
...
```

2.3 Solution #2

Code for this task can be found in task2.py file. The code consists of a single MapReduce step, where a mapper to filters relevant data and a reducer to calculates and output the minimum temperatures.

Running a file (with path to *1800.csv* as an argument) produces following output:

```
"ITE00100554"    -14.8
"EZE00100082"    -13.5
```

2.4 Solution task #3

Code for this task can be found in task3.py file. In the code MapReduce job with two steps is defined: the first step extracts words and counts them, and the second step sorts and outputs the word count

Running a file (with path to *Book.txt* as an argument) produces following output:

```
"the"    212
"and"    140
""       116
"a"      103
"to"     90
"his"    83
"of"     76
"he"     67
....
```

2.5 Solution #4

Code for this task can be found in task4.py file. In the code MapReduce job comprises two steps: the first step extracts customer IDs and spending amounts, and the second step sums and sorts the spending amounts.

Running a file (with path to *customer-orders.csv* as an argument) produces following output:

```
"68"    6375.449999999997
"73"    6206.199999999999
"39"    6193.109999999999
"54"    6065.389999999999
"71"    5995.660000000003
....
```

3 ASSIGNMENT 2

3.1 Goals of assignment

Goals of this assignment were to solve tasks #2, #3, #4 and #5 with Spark RDD.

3.2 Solution #2

Code for this task can be found in task1.py file. The file reads the data, filters out non-"TMIN" entries, and then maps each line to a key-value pair with the city as the key and the corresponding temperature as the value. Afterward, it uses the **reduceByKey** transformation with the min function to find the minimum temperature for each city. Finally, it collects and prints the results:

```
[('ITE00100554', -14.8), ('EZE00100082', -13.5)]
```

3.3 Solution #3

Code for this task can be found in task_2_and_3.py file. The file uses PySpark to read a text file, tokenize the words, count their frequencies, sort the results, and then print the words along with their frequencies in descending order:

```
the : 212
and : 140
a : 103
to : 90
his : 83
of : 76
he : 67
...
```

3.4 Solution #4

Code for this task can be found in task_4_and_5.py file. The script sets up a Spark context, reads the CSV file, defines a function to parse each line, maps the data, performs a reduce operation to sum spending for each customer, and then sorts and prints the results in descending order based on spending:

```
('68', 6375.449999999997)
('73', 6206.199999999999)
('39', 6193.109999999999)
('54', 6065.389999999999)
('71', 5995.660000000003)
('2', 5994.59)
('97', 5977.189999999995)
('46', 5963.109999999999)
('42', 5696.840000000003)
...
```

3.5 Solution #5

Code for this task can be found in task_6_and_7.py file. It sets up a Spark context and reads two files, "Marvel+Graph.txt" and "Marvel+Names.txt.". The script tokenizes the lines in the graph file, counts character appearances, and sorts the results. It determines the most and least popular character IDs from the sorted results then it maps character IDs to names using the names file. Finally names are printed:

```
['859', '"CAPTAIN␣AMERICA" ']
['4602', '"RED␣WOLF␣II" ']
```

4 ASSIGNMENT 3

4.1 Goals of assignment

Goals of this assignment were to solve tasks #2, #3, #4, and #5 with PySpark SQL.

4.2 Solution #2

The PySpark code initializes a Spark session named "SparkSchemaDemo" with a local master and three worker threads. It reads temperature data from a CSV file named "1800.csv" into a DataFrame, selecting and renaming specific columns. The DataFrame is then filtered to include only rows where the weather station name is either 'ITE00100554' or 'EZE00100082' and the observation type is 'TMIN' (minimum temperature). Finally, the minimum temperature for each weather station is calculated and displayed using the groupby and min functions. The output is the following:

```
+-----+-----+
| name|min(temp)|
+-----+-----+
|ITE00100554| -148|
|EZE00100082| -135|
+-----+-----+
```

4.3 Solution #3

The PySpark code creates a Spark session named "WordFrequency" and reads the contents of a text file named "Book.txt" into a DataFrame. It then processes the text data by splitting each line into words, exploding the resulting array of words into individual rows, and counting the frequency of each word. The word frequencies are sorted in descending order. The resulting DataFrame, word_freq, contains each unique word and its corresponding count in the text data. The output is the following:

```
+---+---+
| the| 212|
| and| 140|
| | 116|
| a| 103|
| to| 90|
| his| 83|
| of| 76|
| he| 67|
| was| 66|
| she| 57|
| in| 53|
|that| 42|
| at| 40|
|with| 38|
| had| 38|
| her| 37|
|said| 36|
|Mrs.| 23|
| it| 22|
| as| 22|
+---+---+
only showing top 20 rows
```

4.4 Solution #4

The PySpark code establishes a Spark session named "SparkSchemaDemo" with a local master and three worker threads. It reads data from a CSV file named "customer-orders.csv" into a DataFrame, selecting and renaming specific columns. The code then groups the data by customer ID, calculates the total amount spent by each customer, and sorts the results in descending order based on the total amount. Finally, it displays the aggregated data, showing the customer IDs and their corresponding total amounts spent. The output is the following:

```
+—+————————+
| id| sum(amount)|
+—+————————+
| 68| 6375.449999999997|
| 73| 6206.199999999999|
| 39| 6193.109999999999|
| 54| 6065.389999999999|
| 71| 5995.660000000003|
| 2| 5994.59|
| 97| 5977.189999999995|
| 46| 5963.109999999999|
| 42| 5696.840000000003|
| 59| 5642.89|
| 41| 5637.62|
| 0| 5524.949999999998|
| 8| 5517.240000000001|
| 85| 5503.43|
| 61| 5497.479999999998|
| 32| 5496.050000000004|
| 58|5437.7300000000005|
| 63| 5415.150000000001|
| 15| 5413.510000000001|
| 6| 5397.879999999998|
+—+————————+
only showing top 20 rows
```

4.5 Solution #5

The PySpark code analyzes Marvel superhero data using Spark SQL. It reads two text files, 'Marvel+Graph.txt' containing hero connections and 'Marvel+Names.txt' with hero names and IDs. The hero connections are processed to find the most and least popular superheroes based on their connections. The results, including hero names and IDs, are displayed using Spark DataFrames. The application runs within a Spark session named "WordFrequency." The output for the least popular is the following:

```
+————————+—-+————————+
| value| ID| Name|
+————————+—-+————————+
|2117 "GERVASE, LA...|2117|GERVASE, LADY ALY...|
+————————+—-+————————+
```

The output for the most popular is the following:

```
+————————+—+————————+
| value| ID| Name|
+————————+—+————————+
|859 "CAPTAIN AMER...|859|CAPTAIN AMERICA"|
+————————+—+————————+
```

5 ASSIGNMENT 4

5.1 Goals of assignment

Goals of this assignment were to: Reads text data from a socket, split the lines into words, and calculate the count of each word in real-time using Spark Structured Streaming.

5.2 Solution #1

Code for this task can be found in task1.py file. It reads text data from a specified socket, splits the lines into individual words, and calculates the running count of each word. The results are continuously updated and printed to the console. The application remains active, awaiting new data and updating the word counts in real-time until manually stopped.