

CENG 790: Big Data Analytics, Fall 2020

Assignment 1: Introduction to Spark

This assignment is part of the evaluation of CENG 790 course and it should be done individually. The deadline is 15 November 2020 23:59. The late submissions will receive a penalty of %25 for each day, starting from 16 November 2020 00:00.

You should submit a report containing the answers of the questions for each part. You may provide the screenshots of the output where necessary. You should also submit the code you produced to answer the questions. Make sure that your code is understandable, use comments extensively to explain what code piece is written for which question.

Make sure you follow the [METU Academic Integrity guide](#). Copying code from internet is also considered as cheating. Make sure the code you produced is original.

In the context of this work, we will be processing data originating from the Flickr website. The data file contains meta-data related to pictures that were uploaded on Flickr. We prepared a folder containing a skeleton for the Spark project, together with the necessary files.

Part 1: Processing data using the DataFrame API

The file Part1.scala contains the beginning of a program allowing you to load the Flickr data within a DataFrame. A DataFrame in Spark is a column-based data structure with a schema. DataFrames are very popular because they can be used with SQL queries, just like any RDBMS. Hence, most of this exercise consists in writing SQL queries. In practice, Spark relies on RDDs to implement DataFrames. As the csv file we process does not contain a header, we manually declare the schema of the data. When a header (i.e. a line giving a name to each column) is present, Spark uses it automatically and discovers the type of each column by looking at the data.

1. Using the Spark SQL API (accessible with `spark.sql("...")`), select fields containing the identifier, GPS coordinates, and type of license of each picture.
2. Create a DataFrame containing only data of interesting pictures, i.e. **pictures** for which the **license information is not null**, and GPS coordinates are valid (not -1.0).
3. Display the execution plan used by Spark to compute the content of this DataFrame(`explain()`).
4. Display the data of this pictures (`show()`). Keep in mind that Spark uses lazy execution, so as long as we do not perform any action, the transformations are not executed.
5. Our goal is now to select the pictures whose license is NonDerivative. To this end we will use a second file containing the properties of each license. Load this file in a DataFrame and do a join operation to identify pictures that are both interesting and NonDerivative. Examine the execution plan and display the results.
6. During a work session, it is likely that we reuse multiple time the DataFrame of interesting pictures. I would be a good idea to cache it to avoid recomputing it from the file each time we use it. Do this, and examine the execution plan of the join operation again. What do you notice?
7. Save the final result in a csv file (`write`). Don't forget to add a header to reuse it more easily.

Part 2: Processing data using RDDs

The file Part2.scala contains the beginning of a program loading the Flickr data using a RDD. You are also given a class Picture.scala representing a picture.

1. Display the 5 lines of the RDD (take(5)) and display the number of elements in the RDD (count()).
2. Transform the RDD[String] in RDD[Picture] using the Picture class. Only keep interesting pictures having a valid country and tags. To check your program, display 5 elements.
3. Now group these images by country (groupBy). Print the list of images corresponding to the first country. What is the type of this RDD?
4. We now wish to process an RDD containing pairs in which the first element is a country, and the second element is the list of tags used on pictures taken in this country. When a tag is used on multiple pictures, it should appear multiple times in the list. As each image has its own list of tags, we need to concatenate these lists, and the flatten function could be useful.
5. We wish to avoid repetitions in the list of tags, and would rather like to have each tag associated to its frequency. Hence, we want to build a RDD of type RDD[(Country, Map[String, Int])]. The groupBy(identity) function, equivalent to groupBy(x=>x) could be useful.
6. There are often several ways to obtain a result. The method we used to compute the frequency of tags in each country quickly reaches a state in which the size of the RDD is the number of countries. This can limit the parallelism of the execution as the number of countries is often quite small. Can you propose another way to reach the same result without reducing the size of the RDD until the very end?

HAVE FUN and GOOD LUCK !