✓ Terminé

For this activity, you must write Spark code in Python or Scala.

This activity was adapted from a project organized by David Auber from the University of Bordeaux.

Data from Twitter

The data set we will use came from the archive.org website and can be downloaded free of charge. It contains public messages posted by users of the Twitter platform on March 2020.

The input files can be found at `/user/auber/data_ple/tweets/`. Each line contains information about a single tweet in jsonnl format, which can be easily converted to a Python dictionary. **It is important to notice not all keys are present in all lines.**

```
import json
my_dict = json.loads(my_line)  #my_line is a string in jsonnl
                                                       #format, my_dict will be a
                                                       #dictionary
```

Once you convert each line to a dictionary, the tweet itself can be found in the key `["text"]` and the number of retweets in `[“retweeted_status”]["retweet_count"]`. Information about the author is in a dictionary under the key `["user"]`: `["user"]["screen_name"]` is the username, and `["user"]["followers_count"]` the number of followers.

The hashtags (if present) are available in `["entities"]["hashtags"]`, which is a list of dictionaries, one per hashtag, where each of them will be under the key `["text"]`, without the `#`.

Here is a function to, given a tweet dictionary, return a list of the hashtags (**it supposes the `entities` and `hashtags` keys are present, there will be ugly errors if they are not**):

```
def get_hashtags(tweet):
    tags = []
    assert "entities" in tweet
    assert "hashtags" in tweet["entities"]
    for tag in tweet["entities"]["hashtags"]:
        tags.append("#"+tag["text"])
    return tags
```

The dataset is large so start by working with a smaller version to test your solutions. Smaller versions are available in the HDFS cluster as `/user/fzanonboito/CISD/smaller_twitter.json` (~2.5 GB) and `/user/fzanonboito/CISD/tiny_twitter.json` (~1 GB). You may also benefit from defining a number of executors for your application (instead of the default number, which is 2).

Finally, an important advice is: always make sure to filter out entries that are not relevant. For example, if we were doing statistics on language, we would start by preparing an RDD with only the tweets that have such information in them, and even maybe change the format to keep only the few fields that are relevant for our analysis. It is interesting to notice a considerable amount of data is used by the tweets themselves (the text), but we will never use that in the questions below.

1. Obtain the 20 most popular hashtags.
2. To each hashtag, find the list of all other hashtags that were used in the same tweets, and the number of times they appeared together.

Number of partitions

Conduct a study of the impact of the number of partitions on performance when using Spark. We are particularly interested in answering the following questions.

1. What is the default number of partitions when creating an RDD from a file? Does it depend on the number of executors? And on the size of the file? Is it related to the HDFS chunk size (128 MB on LSD)?

2. A commonly found advice online is to use between 1 and 3 partitions per executor core (i.e. if using N executors, each with M cores, we would make sure our RDD has NM, 2NM, or 3N*M partitions) to maximize parallelism. Is it true? Is it true for any file size? You may change the number of cores per executor with the `--executor-cores` option, given to spark-submit.
3. We can imagine that, after applying a filter, or a map where we dramatically change the size of each element, partitions would have different sizes and some of them would become extremely small. In this case, is it useful to repartition the obtained RDD before proceeding (i.e. what is the cost of doing so vs. how much faster subsequent operations will be)?
4. Based on your conclusions from this study, what would you recommend?

Start by studying the RDD documentation: there are a number of functions that allow for changing the number of partitions of RDDs (they are in fact transformation that create new RDDs), and many transformations (such as `map` and `reduceByKey`, even notably `textfile`) have optional parameters that control the number of partitions of the resulting RDD.

Remember that data sets of different sizes are available. Comparing results observed with different sizes may help you answer the questions.

Also, beware of lazy evaluation - if you don't apply an **action** to the RDD (such as `count` or `saveAsTextFile`), it may not be generated.

Report

For the second report, which will make for 30% of your grade for this course, you are asked to write about this activity (TD 4). You are asked to:

- present and explain your solution in details.
- Explain your study about the number of partitions in details: what experiments you designed to answer your questions and why, your experimental methodology, the obtained results, your conclusions from them.
- Respect a **page count limit of 5** and submit your **report in .pdf format, accompanied by your source code (in a py or scala file).** Keep in mind that the study of the impact of the number of partitions on performance will account for most of the grade, so you should use the available space accordingly.
- Include the obtained results for the given file. That can be done as an appendix and does not count for the page count limitation. As long as not excessive, figures and code snippets may also be added to an appendix.
- **Work alone or in groups of 2.**
- Not copy anything from the internet or from colleagues, to not ask for solutions in online forums, and to cite all external sources you use.
- Write in French or English (or Portuguese :).

Grades will be based on:

- quality of the report: readability, clarity, organization, etc;
- scientific rigor applied to the study, including but not limited to experimental methodology;
- correctness of the provided arguments and explanations;
- correctness and performance of the provided solution.

Modifié le: Friday 2 December 2022, 11:42