Lab 2 - Spark Batch Applications on ElasticMapReduce

Jesus Omaña Iglesias & Pablo Castagnaro

Large Scale Distributed Systems - 2023/24 Edition

0.1 Changelog

2024/02/06

1 Objective of this lab

- to implement a language filter similar to Lab 1 using Apache Spark
- to run the application on AWS ElasticMapReduce through the AWS Web console and benchmark it
- to enhance the SimplifiedTweet class to access additional fields
- to create more complex applications using Apache Spark

2 Implement the Twitter filter using Spark (2.5 points)

In this section, you will re-implement the `TwitterLanguageFilter` using the Spark framework. The objective is the same as in the previous Lab: to select all tweets in a given language and store them in S3.

You should use the provided `WordCount` example as starting project for your application.

Add the corresponding packages (edu.upf) and start coding!

At the end of this practise, your project will contain more than one executable class. That's okay! Use the `spark-submit` options to select the class that you want to execute each time!

2.1 Additional Remarks

- Please name your main class for this application `TwitterLanguageFilterApp`. Put it in a package named `edu.upf` to keep a good modularity in your project.
- In this exercise, you should re-use Lab1's class `SimplifiedTweet`. Put it in a package named `edu.upf.model`.
- Your application should be runnable using the following order of parameters:

  ```
  spark-submit --master <YOUR MASTER> --class <YOUR MAIN CLASS> your.jar <language> <output> <inputFile/Folder>
  ```

- The Spark's function `sparkContext.textFile(path:String)` allows you to use a single file or a folder, in the latter case, it will read all the files inside of the specified folder and return a `RDD<String>` with the content of all of them concatenated.

3 Benchmark the Spark-based TwitterFilter application on EMR (2.5 points)

Important

When you launch a cluster, you start spending AWS credit! Remember to terminate your cluster at the end of your experiments!

In this exercise, you'll run the application in AWS using the ElasticMapReduce service and creating a real Hadoop/Spark cluster. You should:

- Run the same experiments as in Lab1: gathering all the tweets in Spanish (`es`), English (`en`) and Catalan (`ca`) languages.
- Place the result of such runs in the appropriate location in S3, as described below
- Document the elapsed time (as seen in the AWS console) in the README.md within your repo.

3.1 Running Spark jobs on EMR

Running on EMR is not a complex task, but needs some preparation. Each student should run the following steps:

- load your input onto S3
- create a cluster supporting Spark within EMR (see Lab2 slides in Moodle)
- load your jar (with its dependencies) on S3
- create a Spark step in the AWS console, which will execute your application

3.1.1 Load input/jars onto s3

Refer to Lab2 slides for hints. Additionally:

- create a bucket named: `lsds2024.lab2.output.<USER-ID>`, where `<USER-ID>` should be changed for your user identifier within the UPF (the number starting with U and followed by digits, like `Uxxxxxx`
- keep the bucket private (it's the default setting when creating a bucket), but give access to your professor. The canonical id for the accounts we are using are:
  - **Luca:** `7b46381f5e75fcd7ba5f3f2076d98d99fcfe757623eee263d452766f1d22ff1b`
  - **Pablo:** `9fd903100fcd1c2e1502f89e592a20e5cf3caf63b3fafa2f43c80c7397e97b42`
  - **Academy:** `badcde72817c72af38dfde8186a86029f6c6e726c598a81bdb7ec8ef5e2fa041`

  they are also available in Moodle for a more convenient C&P. Refer to the slides for how to give access to you professor's account.
- keep the following structure within your bucket:

```
| your-bucket
\___input
|  \___your input file(s)
\___jars
```

```
|  \___your jar file(s)
\___output
   \___ benchmark
       \___ es
       \___ hu
       \___ pt
   \___ run1
   \___ run2
   \___ ...
   \___ runX a output directory used in execution X (you can use it for testing, as you
       might want/need more than one execution before you achieve success)
```

3.2 Additional Remarks

- The input of your application must be the entire collection of tweets, as in the previous lab. Upload it to the appropriate location on S3 as soon as possible, as it might take some time!
- The output should be in a directory specified as command line argument. If executing locally, this can be a directory on your workstation; when executing on an EMR cluster, one of the biggest advantages of Spark, is that it can automatically manage r/w access to S3.
- To run locally, your master should be set as `local` (you can also omit completely the –master parameter when running locally)
- Remember to use the directory `output/benchmark` to store the results for each language!

4 Most popular bi-grams in a given language (2.5 points)

In this exercise, you'll write a Spark application named `BiGramsApp` that answers the following question:

> What are the top-10 most popular bi-grams for a given language from *original* Tweets used during the 2018 Eurovision Festival?

Bi-grams are defined as a pair of consecutive written units. We will be using words as text units. For instance, the sentence "The cat is on the table", produces the following sequence of bi-grams: `<the, cat>`, `<cat, is>`, `<is, on>`, `<on, the>`, `<the, table>`.

You want to calculate bigrams *for a given language* from each `SimplifiedTweet` text and then do the counting of each pair. For this exercise, the *word count* exercise presented in seminars is a good starting point.

4.1 Parsing "original" Tweets

We define original tweets as *tweets that are not retweeted*. To process this information we'll create a class, named `ExtendedSimplifiedTweet`, similar to the original `SimplifiedTweet`, but with additional fields. This class should also used for the rest of the exercises.

Use the following definition to implement the class:

Partial model for a simplified tweet

```
public class ExtendedSimplifiedTweet implements Serializable {

  private final long tweetId;    // the id of the tweet ('id')
  private final String text;        // the content of the tweet ('text')
  private final long userId;        // the user id ('user->id')
```

```
private final String userName;  // the user name ('user'->'name')
private final long followersCount; // the number of followers ('user'->'followers_count')
private final String language;  // the language of a tweet ('lang')
private final boolean isRetweeted; // is it a retweet? (the object 'retweeted_status' exists?)
private final Long retweetedUserId; // [if retweeted] ('retweeted_status'->'user'->'id')
private final Long retweetedTweetId; // [if retweeted] ('retweeted_status'->'id')
private final long timestampMs;  // seconds from epoch ('timestamp_ms')

public ExtendedSimplifiedTweet(long tweetId, String text, long userId, String userName,
                               long followersCount, String language, boolean isRetweeted,
                               Long retweetedUserId, Long retweetedTweetId, long timestampMs) {
 // IMPLEMENT ME
}

/**
 * Returns a {@link ExtendedSimplifiedTweet} from a JSON String.
 * If parsing fails, for any reason, return an {@link Optional#empty()}
 *
 * @param jsonStr
 * @return an {@link Optional} of a {@link ExtendedSimplifiedTweet}
 */
public static Optional<ExtendedSimplifiedTweet> fromJson(String jsonStr) {
 // IMPLEMENT ME
}
```

4.2 Additional remarks

- Please name your main class for this application `BiGramsApp`. Put it in a package named `spark`.
- Please write the obtained results inside the "README.md" file.
- Dealing with text can be complicated, due to language peculiarities. To make the exercise more meaningful, you should "normalise" (see example wordcount application on Moodle) words applying at least the following:

  word trimming (no whitespace at beginning or end of word)

  lower-casing (to avoid considering "Eurovision" and "eurovision" as two different words)

  filter empty words
- Make sure that your application uses the `ExtendedSimplifiedTweet` class inside the `model` package.
- Depending on your implementation, you might require an additional class to handle the Bigrams. It should store the two words of a given bigram and override the function `equals()` to make proper comparison between Bigram objects
- It's enough if your application saves the all the results sorted descending by number of appearances and you just include in the README.md file the top 10 entries.
- Your application should be runnable using the following order of parameters:

  `spark-submit --master <YOUR MASTER> --class <YOUR MAIN CLASS> your.jar <language> <output> <inputFile/Folder>`

5 Most Retweeted Tweets for Most Retweeted Users (2.5 points)

In this last exercise, you should find the most retweeted tweet for the 10 most retweeted users (one tweet per user, the most retweeted for that user).

You'll want to do this exercise in stages, first finding the most retweeted users (hint: use the retweeted-user-id to identify such users), then finding the most retweeted tweets, then using both parts to produce the final answer.

Be aware that certain users might be very retweeted and still not have tweets in this dataset (their tweets might have appeared "earlier" in time, so they're not part of this dataset).

5.1 Additional remarks

- Please name your main class for this application `MostRetweetedApp`
- Please write the output of this exercise also inside the "README.md" file
- Again, make sure that your application uses the `ExtendedSimplifiedTweet` class
- Your application should be runnable using the following order of parameters (we do not use the language parameter for this exercise):

  ```
  spark-submit --master <YOUR MASTER> --class <YOUR MAIN CLASS> your.jar <output> <inputFile/F
  ```

6 Final remarks. Important!

- Your code should compile. Non-compiling code won't be evaluated
- Your code should be able to create a jar containing all your non-test classes (`mvn package` should work)
- All exercises should be runnable, that is, they should work with `spark-submit`
- All your main classes should be in package `edu.upf`, that is, the `--class` option for spark-submit should have value `edu.upf.<name-of-the-class>`
- Submit your final code in Moodle (one submission per group) as a .zip file containing your code

7 Deadlines

- deadline Feb, 28th at 23:59