387 - Lab 7 - Spark

Dealing with massive amounts of data often requires parallelization and cluster computing; Apache Spark is an industry-standard for doing just that. In this lab, we introduce the basics of PySpark, Spark's Python API for this. Apache Spark is an open-source, general-purpose distributed computing system used for big data analytics. Spark is able to complete jobs substantially faster than previous big data tools (i.e. Apache Hadoop) because of its in-memory caching and optimized query execution. Spark provides development APIs in Python, Java, Scala, and R. On top of the main computing framework, Spark provides machine learning, SQL, graph analysis, and streaming libraries.

In this lab, you will get to explore a bit of Apache Spark. In particular, you will learn about transforming RDDs and actions on RDDs as well as manipulating DataFrames in PySpark.

Installation:

- Check if Java 11 is installed
  ```
  java -version
  ```
  If not install by using the below commands or any other way you wish
  ```
  sudo apt update
  sudo apt install default-jdk
  java -version
  ```

- Now install pyspark
  ```
  pip3 install pyspark
  ```

PART 1 Word Count:

You have to process a collection of JSON files (created from news websites) to do word counts. A zip file containing multiple input JSON files can be downloaded from Moodle. Each input file is a JSON file, which has several attributes; the ones we are interested in counting the words are these: *date_published, article_body*

First, unzip the input files into a directory. You have to write a program to find the word count in the *article_body* attribute in each (year-month) (e.g. 2018-06, 2018-07, etc., which can be extracted from *date_published*); the output would be of the form ('year-month word', count).

You will use the **map and reduce** paradigm to carry out this task. Treat the below code as a skeleton and fill it up wherever asked. Please note that 'year-month word' is referred to as a word.

```
from pyspark.sql import SQLContext
```

```
from pyspark import SparkContext
# other required imports here

def process(line):
    # fill
    # convert all characters into lower case
    # replace all non-alphanumerics with whitespace
    # split on whitespaces
    # return list of words

if __name__ == "__main__":
    # create Spark context with necessary configuration
    spark = SparkContext("local", "Word Count")

    # read json data from the newdata directory
    df = SQLContext(spark).read.option("multiLine", True) \
    .option("mode", "PERMISSIVE").json("./newsdata")

    # split each line into words
    lines = df.select("date_published", "article_body").rdd
    words = lines.flatMap(process)

    # count the occurrence of each word
    wordCounts = words.map(#fill).reduceByKey(#fill)

    # save the counts to output
    wordCounts.saveAsTextFile("./wordcount/")
```

You can then run the above program using

```
#> spark-submit wordcount.py
```

PART 2 Stock Returns:

The file stock_prices.csv contains the daily opening and closing price of a few stocks on the NYSE/NASDAQ. Compute the average daily return of all stock for each day. Return for a stock on a day is computed as the percentage difference between closing and opening prices. Output schema should be (date, avg_return).

Save the output dataframe using `write.csv(‘./stockreturns/’)`. You need not print anything on the screen.

You should use only Data Frames API to solve this part (part 2). Do not use RDDs.

PART 3 Web Crawling:

In this part, you have to build a web crawler using Spark. For simplicity, you can restrict your crawl to a dummy website we created, starting from *here*. You will only follow relative URLs (without a protocol specifier or hostname), which point into the same site, not external URLs.

In each round, you have a dataset of URLs to crawl and generate a new dataset of URLs to be crawled in the next round, as detailed below.

In a round, you use the dataset with the given URLs, use flatmap to download the page content from each URL and then extract all relative local URLs from the downloaded page. To do so, you look for strings of the form <a href="xyz"> where "xyz" does not start with http. Also, when you download a page, first check the content type, and ignore it if the content type is not html. There are many documents, image and pdf files which you really don't want to download. You have to include such URLs in the output, but you don’t crawl them. Here's how you can do that:

```
h = requests.head(url)
header = h.headers
content_type = header.get('content-type')
```

You define a function that takes a URL and returns an iterator on the list of URLs found in that URL and provide this function to the flatmap function.

Now you will need to iterate on top of the above. In each round, you get URLs as above, and union/except operations on datasets to track the set of all (relative) URLs found so far, and the new ones found in this round. You need to force execution by caching the dataset and doing a count on it. If the new one is a non-empty set (count > 0), you continue iterating. Otherwise, you output the dataset of all relative URLs along with their indegree.

The output would be tuples of (relative_url, indegree). relative_url should start with ‘/’, and if it’s an HTML page it should end with ‘.html’. Here, indegree of a URL means the no of URLs pointing to it. For the relative URL /1.html, it would be (no of URLs pointing to it + 1) as we are starting with it. You need not print anything on the screen. Save the output using
`saveAsTextFile(‘./webcrawler/’)`

You should only use RDDs and transformations and actions on them to solve this part.

Resources:

- Dataframes
- RDD
- http://users.csc.calpoly.edu/~dekhtyar/369-Winter2019/papers/pyspark.pdf
- https://spark.apache.org/docs/1.6.1/sql-programming-guide.html

Submission Instructions:

- Make a folder named **<rollnumber1>-<rollnumbe2>-a7** containing **wordcount.py**, **stockreturns.py** and **webcrawler.py**.
- Your submission should not contain any other files
- **Zip** the folder and upload it to the **moodle**
- Before submitting, check all the file names and run your code one more time to ensure that everything is working fine as we'll only run the python files described above via a script and ANY naming errors will result in you not receiving any marks.

Grading Rubric

Exercise	Marks
1	10
2	15
3	25
Total	50