Language Abstractions for Concurrent and Parallel Programming 2021 Assignment 3: MapReduce / Spark

Prepared by Georgios Panayiotou

Due Date: Monday, January 10, 2022 at 17:00.

Lab Date: Wednesday, December 15, 2021
The assignment can be done in pairs (within your project groups). In that case, a submission from either of the group members suffices.

Introduction

Suppose you are working for an online retail store and have been tasked with generating reports summarizing the year's sales. Because the yearly sales data and product catalogue are both large, you will need to use MapReduce and Spark to produce the reports.

Data Sets

The sales data is broken into three datasets, which will be provided to you as `.csv` files in a compressed folder. Each row/line of the `.csv` file corresponds to a data point, and the different columns are separated by commas. The first line is possibly a header.

The **Categories** dataset describes the hierarchy of item categories sold by the online store. This hierarchy contains top-level categories such as `Clothes`, which may be broken into subcategories, such as `Mens Clothes` and `Womens Clothes`, which in turn may be broken into further subcategories, such as `Shirts`, `Shorts`, `Jeans`, and so forth. Categories that have no subcategories are *leaf categories*. Only leaf categories have items associated with them. The hierarchy is not guaranteed to be balanced in any way nor be of any fixed depth. The corresponding file, `categories.csv`, will look similar to the following. The type of each column is string.

```
Category,Subcategory
All,Books
All,Clothes
All,Electrical
All,White Goods
Books,Kids
Books,Literature
Books,Non-Fiction
```

The first line is a header and should not be included in your processing. It can be used as a header for a DataFrame.

The **ItemCategories** dataset gives every item sold in the online store along with the leaf category it belongs to. Items will be associated with precisely one leaf category, though via the entries in the **Categories** dataset it will belong to other (super)categories. The corresponding file, `item_categories.csv`, will look like the following. The type of each column is string.

```
Item Id,Category
337334,Kids
766545,Kids
134739,Kids
796624,Kids
703296,Kids
310651,Literature
114277,Literature
```

Again, the first line is a header.

The **Sales** dataset captures all sales made by the online store. Each entry includes the date of the sale, the id of item sold, and the amount the item was sold for. The file corresponding file, `sales.csv`, will look like the following. The first column is the date/time of sales (of type datetime), the second column is the item ids (type string), and the final column is the amount of the sales in kronor (type integer).

```
2017-01-01 00:01,101166,969
2017-01-01 00:01,107806,3738
2017-01-01 00:01,197623,381
2017-01-01 00:02,141985,115
2017-01-01 00:02,248813,131
2017-01-01 00:03,107806,3738
2017-01-01 00:03,141985,115
2017-01-01 00:03,151908,1115
2017-01-01 00:04,107806,3738
2017-01-01 00:04,141985,115
2017-01-01 00:04,197623,381
```

The sales dataset does **not** have a header.

Note that the datasets provided for you to develop your solutions may not be the same as the datasets used for testing your solutions, so do not make your code depend on specific values found in the datasets. For example, there may be a different hierarchy of item categories.

The Problems

The objective of this assignment is to perform three analyses on the dataset using MapReduce, Spark's RDDs, and Spark's DataFrames — one analysis will be performed using each framework. In each case, the Python programming language and Pyspark, Python's Spark installation, must be used. Instructions for installing Pyspark are provided below. The MapReduce framework to be used is one provided for you built upon Pyspark. Using this framework avoids the hassle of installing and configuring Hadoop.

Warm Up Rather than jumping straight into the questions below, in order to help you get started, try to compute the following in each framework: the total of all sales; the sales per leaf category. *This question does not need to be handed in, it is not assessed.*

Question 1: MapReduce

Given the sales data, produce a report that summarises the sales data for the whole year in categories. This means that you should produce a report such as the following—the precise format is unimportant.

```
Category             Total Sales
---------------------------------------
ALL                   134666
  shoes                  33354
    sneakers               32123
    sandals                 1231
  cups                     1012
  cars                   100300
```

Here `sandals`, for example, is a leaf category and the amount associated with it is computed by totalling all the sales for all items in that category, as indicated by the **ItemCategories** dataset. The value associated with non-leaf categories, such as `shoes`, is the sum of the totals for each immediate subcategory. Thus the total associated with `shoes` is the sum of the totals for `sneakers` and `sandals`, assuming that `sneakers` and `sandals` are the only subcategories of `shoes`. Ultimately, the total sales for the root category (if there is a root category) will be the total sales over all.

For this question the Sales data must be processed using the MapReduce framework provided. Data in the **Categories** and **ItemCategory** data sets can be processed using regular Python.

Question 2: Spark RDDs

The objective of this question is to produce a a sales report similar to that of the previous question, except that all totals are broken down into weekday and weekend sales, as indicated by the following example output (again the format is only a suggestion).[1]

```
Category             Weekday      Weekend
-----------------------------------------
ALL                   134566          100
```

[1] See `https://stackoverflow.com/questions/9847213/how-do-i-get-the-day-of-week-given-a-date-in-python`

```
shoes                  33344          10
  sneakers               32118           5
  sandals                 1226           5
cups                      972          40
cars                    100250          50
```

For this question, all computation on sales must be done using Spark RDDs and the operations available on them. To get you started, use the following code to bring the sales data into an RDD:

```
from pyspark import SparkContext
sc = SparkContext("local", "Simple App")
input = sc.textFile("sales.csv")
```

Question 3: Spark DataFrames

The objective of this question is to compute another sales report, but with two changes. Firstly, the category **Clothes** and any subcategories of it and any items of those categories must not be included in the report. Secondly, the report should present sales per month, as indicated below:

```
+-------------+---------+---------+---------+---------+---------+---------+--------
|     Category|  January| February|    March|    April|      May|     June|    ...
+-------------+---------+---------+---------+---------+---------+---------+--------
|Front Loaders| 31341302| 27650877| 25796509| 30623732| 29850540| 32362868|    ...
|    Computers|   946853|   886049|   722641|   842406|  1095324|  1171518|    ...
|        Books| 64086657| 54956734| 55111569| 67193968| 58060805| 56586136|    ...
|  Top Loaders| 36272382| 31607272| 31458459| 34312841| 27463432| 25685998|    ...
|  White Goods|146360518|127907711|127156909|144582048|129053200|126514192|    ...
|          ...|      ...|      ...|      ...|      ...|      ...|      ...|    ...
+-------------+---------+---------+---------+---------+---------+---------+--------
```

For this question, computations on all data sets must be done using Spark DataFrames and the operations available on them. To get you started, use the following code to bring the sales data into an DataFrame:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import TimestampType, StringType,\
                        IntegerType, StructType, StructField

spark = SparkSession.builder.getOrCreate()

schema = StructType([StructField("Time", TimestampType()), \
                     StructField("Item Id", StringType()), \
                     StructField("Price", IntegerType())])

sales = spark.read.csv('sales.csv', header = False, \
                        timestampFormat='yyyy-MM-dd HH:mm', schema=schema)
```

The value in variable `schema` ensures that the columns in the data set are converted to the correct types. The default type for a column is string, so reading in the categories and item categories datasets will be simpler.

Installing Spark

This assignment will use Apache Spark and Python, via a package called `pyspark`. Unfortunately, Spark has not yet been installed on the departmental computers. Installing it is quite easy.

Firstly install or possibly upgrade Python, if you haven't done so already.

Next, try either of the following to install pyspark:

```
pip install pyspark
```

or

```
pip3 install pyspark
```

depending on your version of Python.

You can now test whether your installation works by running the shell:

```
pyspark
```

NOTE: You do not need to install Hadoop. Installing Hadoop will take a lot of effort to configure. Spark should work without it.

NOTE 2: An Ubuntu-based virtual machine image will be provided in cases where the installation becomes difficult; check the assignment's description on Studium for the download link. Note that the download will be fairly large (about 10GB).

A MapReduce Framework

To keep the overhead of doing this assignment as low as possible, instead of using both Hadoop and Spark, you will instead use a simple MapReduce framework written in Python for part of the assignment. The framework plus some additional utility functions and an example are provided (Figure 1). (The relevant files are provided on Studium.)

The MapReduce framework closely follows the framework described in class. To classes need to be created to plug into the framework, one doing the mapping and one for reducing. The operation of the whole framework is packaged into a single function called `transform`. This function takes an RDD of key-value pairs and two classes as parameters. The Mapper class has a single method called `map`, to be provided by a subclass, that takes a single pair consisting of a key and a value and returns a list of key-value pairs, where the types of the output keys and values are potentially different from the input types. *The framework will apply this function to all key-value pairs in the RDD.*

The Reducer class has a single function called `reduce`, to be provided by a subclass, that takes a single pair of a key and a list of values (of the type produced by map). *The list of values for each key is a list of all the values returned for that key by all mappers in the map phase.* Based on this input, a single key-value pair is returned. *The framework will apply this function to the lists of values associated with each key produced in the mapper.*

The framework also provides two additional functions `initialise` and `finalise`: Function `initialise` reads in and preprocesses the contents of an input file and stores the results into an RDD. The function takes a preprocess function as an argument, and applies this function to every line of the input file. Function `finalise` outputs the contents of the RDD into a textfile.

The following example (Figure 2) is a complete worked example, which takes a text document and does performs a word count using MapReduce. The user of the framework provides three things: a function to pass to `initialise`, a Mapper class and a Reducer class.

```
# next two lines are required if your default Java is > 1.8
# the precise location depends on your machine
import os
os.environ['JAVA_HOME']='/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home'

from pyspark import SparkContext

sc = SparkContext("local", "Simple App")

## MapReduce Framework
def initialise(sc, inputFile, prepare):
    """Open a file and apply the prepare function to each line"""
    input = sc.textFile(inputFile)
    return input.map(prepare)

def finalise(data, outputFile):
    """store data in given file"""
    data.saveAsTextFile(outputFile)

class Mapper:
    def __init__(self):
        self.out = []

    # call in subclass to output a key-value pair
    def emit(self, key, val):
        self.out.append((key, val))

    def result(self):
        return self.out

class Reducer:
    def __init__(self):
        self.out = []

    # call in subclass to output a key-value pair
    def emit(self, key, val):
        self.out.append((key, val))

    def result(self):
        return self.out

def transform(input, mapper, reducer):
    return input.flatMap(lambda kd: mapper().map(kd[0], kd[1]).result()) \
                .groupByKey() \
                .flatMap(lambda kd: reducer().reduce(kd[0], kd[1]).result())
```

Figure 1: Basic MapReduce Framework in Python Example

```
class WCMapper(Mapper):
    # required
    def __init__(self):
        super().__init__()

    # provided by YOU
    def map(self, key, data):
        for x in data.split(" "):
            self.emit(x, 1)
        return self

class WCReducer(Reducer):
    # required
    def __init__(self):
        super().__init__()

    # provided by YOU
    def reduce(self, key, datalist):
        self.emit(key, sum(datalist))
        return self

def wordcount(sc, inputFile, outputFile):
    rdd = initialise(sc, inputFile, lambda line: ("NoKey", line))
    result = transform(rdd, WCMapper, WCReducer)
        # passing class NOT object (which would be WCMapper() etc)
    finalise(result, outputFile)

wordcount(sc, "text.txt", "count.out")
    # will not work if count.out already exists
```

Figure 2: Word Count example using MapReduce Framework

What to Submit

Submit the following via the Student Portal:

- Your source code.
- Instructions describing how to run your code, specifically including any constraints that might make it difficult to run on someone else's machine.
- Short descriptions of how you solved each of the problems above (Questions 1–3).
- A small test case that could *by hand* be used to demonstrate that your code is correct — this should include files `categories.csv`, `item_categories.csv`, and `sales.csv` in a directory called `test_data`.

All textual answers (e.g., instructions, descriptions) must be submitted in a single file in PDF format.

Evaluation Criteria

The most important aspect of the assignment is your ability to phrase solutions to problems in terms of 1) the MapReduce abstraction 2) core Spark RDD operations, and 3) Spark DataFrame operations. Less important are the surrounding bits of code needed to load your data, output your results correctly, and so forth.

Each question is worth **30 points** based on the following criteria:

- correctness of your solution with respect to the specification (correct entries are filtered, data broken into categories/dates appropriately) **8 points**
- code uses the appropriate framework (MapReduce, RDDs, DataFrames) *and* does as much as possible of the computation using that framework, in particular, all computations on the sales data (excluding reading the data in from file) must use the framework **4 points**
- code is generic in the contents of the categories, item categories, and sales **4 points**
- clarity/elegance of your solution **5 points**
- quality and appropriateness of test data **4 points**
- clarity, conciseness and completeness of the report **5 points**

The final assignment score (on a 0-10 scale) is the sum of all three questions' scores, divided by 9.

Good luck!