Intro to Cloud Computing

Brian DelRocini, Rizwan Chowdhury, Kinjal Patel

Report Recitation 1

We performed this recitation using VS Code Jupyter Lab, Google Colab, and PySpark. We did not use Hadoop file system for this recitation, only the local filesystem with Spark.

Some of the problems have their outputs in labeled folders in the solution directory while others have output printed in the Jupyter Notebook. The solutions in the folders are the result of using the saveAsTextFile() method of the RDDs and the solution is split into different files based on partitioning. The solutions that have outputs printed in the notebook will also have screen shots below.

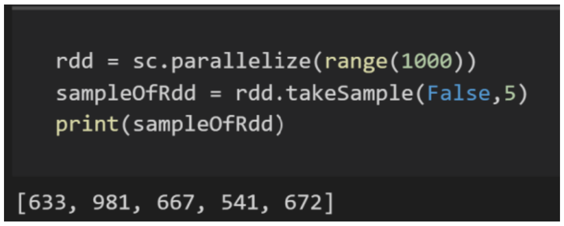
Note: All output files/folder will be within the Outputs folder

We will split the rest of the report by problems.

Problem 1 & 2:

We were able to set up the basic Spark environment and implement the code for these problems without any issue. Problem 2's solution was outputted through the saveAsTextFile() method and can be found in the directory under "Problem02_wc_out"

For Problem 1:

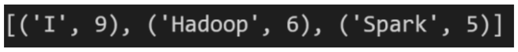


```
rdd = sc.parallelize(range(1000))
sampleOfRdd = rdd.takeSample(False,5)
print(sampleOfRdd)

[633, 981, 667, 541, 672]
```

Problem 3:

We solved problem three by getting all of the words in one list using flatMap then processing the words. Each word's frequency was recorded using map and sortByKey. Then we used the sort() operation using the value instead of the key as the target for comparisons. By doing this, we obtained the most frequent words in "file01_Hd_Sp_Freq", which included terms like "I" which can usually be considered as noise words. What was interesting was the flexibility offered by the Spark sortby() function. By being able to define a specific lambda function for the task we were able to sort by value instead of the key, a task that would have required more round-about methods with MapReduce. We obtained the following output for Problem 3:

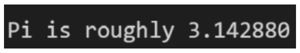


```
[('I', 9), ('Hadoop', 6), ('Spark', 5)]
```

This is from the “file01_Hd_Sp_Freq”, and we verified these results through inspection of the document.

Problem 4:

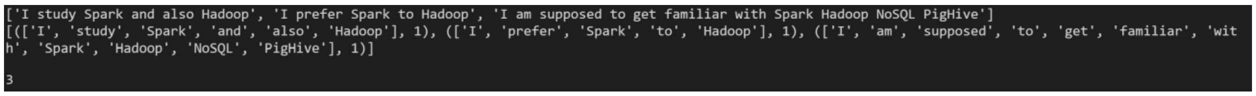
We copied and pasted the demo code and used 10000 samples to get a value around 3.14 for pi. We did notice that a greater number of samples correlated with a value closer to pi. We obtained the following result:



```
Pi is roughly 3.142880
```

Problem 5:

For this problem we utilized the filter operation and Python’s startswith() method to find all lines that start with “I”. Then we used map to tokenize each line into their terms. After that, filter and count were used to get the total occurrences of the term “Spark”. Our result for the occurrence of Spark was 3 which we verified by inspecting the document (“file01_Hd_Sp_Freq”).

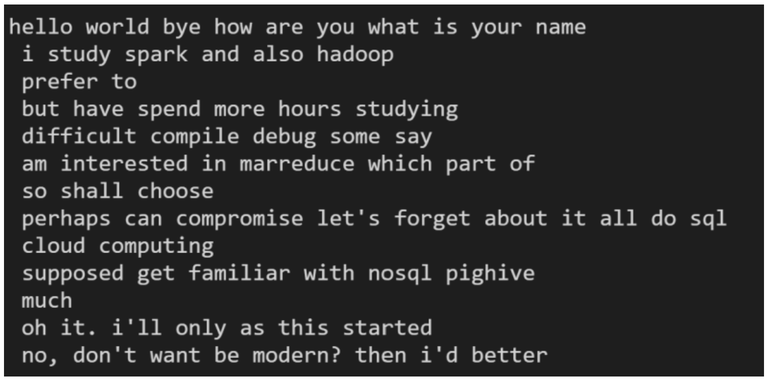


```
['I study Spark and also Hadoop', 'I prefer Spark to Hadoop', 'I am supposed to get familiar with Spark Hadoop NoSQL PigHive']
[(['I', 'study', 'Spark', 'and', 'also', 'Hadoop'], 1), (['I', 'prefer', 'Spark', 'to', 'Hadoop'], 1), (['I', 'am', 'supposed', 'to', 'get', 'familiar', 'wit
h', 'Spark', 'Hadoop', 'NoSQL', 'PigHive'], 1)]

3
```

Problem 6:

We implemented the algorithm for finding and removing duplicates slightly differently for this recitation than we did for the pen and pencil solution. This was necessary due to practical reasons arising from the use of python and spark rather than pseudocode. For every word we created a tuple that was (word,(line,inLine)). “line” is the line in which the word appears and “inLine” is index of the word (by words) inside of the line. We collected all of the (line,inLine) pairs for each word and went through the list for a given word keeping the tuples that have the lowest "line" value. After that we kept only the element with the lowest "inLine" value. Then return the word with firstLine and firstInLinePosition. Once this process was finished, we were left with the first occurrence of each word and its position in the document, the line where it was and the position within that line. Using this information, we re-created the document free of duplicates and with words retaining their original positions relative to each other. A more detailed breakdown of the procedure and methods are in the comments within the quote. The output for this code for the “file01_Hd_Sp_Freq” is below (from notebook) and in the “Problem6_NoDuplicates_Output” folder within the directory.



```
hello world bye how are you what is your name
 i study spark and also hadoop
 prefer to
 but have spend more hours studying
 difficult compile debug some say
 am interested in marreduce which part of
 so shall choose
 perhaps can compromise let's forget about it all do sql
 cloud computing
 supposed get familiar with nosql pighive
 much
 oh it. i'll only as this started
 no, don't want be modern? then i'd better
```

We processed the document line by line and thus recreated it line by line as well.

Problem 7:

We implemented the algorithm for finding and removing duplicates slightly differently for this recitation than we did for the pen and pencil solution. This was necessary due to practical reasons arising from the use of python and spark rather than pseudocode. For the problem, we used flatMap to obtain all words in one list. Then used map and reduce by key to get occurrence of each word. Then we processed each word by its first letter and then get a list of words for each letter. Finally, that list was sorted by the occurrence of the words and return the top 40 (or as many as possible if less) for each letter. More detailed notes for each method and the procedure is in the comments of the code. The output for Problem 7 can be found in both the “Problem07_Top40WordsPerLetter_Ouput” folder and the “Problem07_Ouput.txt” text file. The folder contains the results from the RDD.saveAsTestFile() method and is separated into partitions. The text file contains the output from RDD.collect() method which has been formatted to look nicer.