The Capabilities and Limitations of Hadoop's Parallel Processing: A Qualitative Review Generated Through the Development of Word Counter Tools

Tan Chun Kit
UTM Razak Faculty (UTMRAZAK)
UTM Kuala Lumpur
c.kit@graduate.utm.my

Abstract**—**This paper intended to explore the Hadoop's parallel processing ecosystem, namely the MapReduce and Apache Pig, its capabilities as well as limitations in terms of coding capabilities, general development timeframe, run-time differences and most importantly, its big data processing capabilities. A development of two separate word counter tools was implemented to record and review Hadoop from a down-to-earth perspective. The result suggested there are no significant run-time differences between Python Streamed MapReduce, Tez enabled Apache Pig and Tez disabled Apache Pig. However, the workload and timeframe of developing an Apache Pig equivalent MapReduce Word Counter is significantly longer and far more tedious. The limitations of both MapReduce based and Pig based word counter are discussed.

Keywords— **Big Data Processing, Hadoop, MapReduce, Apache Pig, Word Counter**

I. Introduction

Parallel processing/computing is a computation method in which multiple calculations can be executed simultaneously [1]. This concept received a broad interest due to its scalability and being able to bypass the hardware limitation from serial processing. Due to the upraising of big data, serial processing, and traditional database management systems (DBMSs) slowly become incapable of solving big data questions due to hardware limitations [2]. Big data analysts must choose between purchasing high-performance computing hardware (super-computer) or a commodity server that is capable of parallelly distribute and process huge data. Compare to the former, many fancies the latter mainly due to its scalability and requires no expert knowledge in complicated super-computing.

Hadoop is an open-source distributed processing framework, had received a good amount of recognition in both academic and commercial settings [3]–[6]. The MapReduce and Apache Pig are Hadoop's features in parallel batch processing data from multiple clusters of machines designated through HDFS [7], [8] By scaling up the clusters, it is possible to achieve low processing time when dealing with big data [9]. Both MapReduce and Apache Pig do not require unique coding skills for parallel processing, instead, a general code written in Python and Pig Latin can be streamed using Hadoop Streaming service and the processing framework will deal with the remaining in terms of task distribution and result aggregation.

Despite the fame Hadoop had achieved, there are many drawbacks at the same time. DataFlair Team [10] compiled and listed 13 limitations of Hadoop, these include an issue with small files, slow processing speed, no delta iteration, and low abstraction as well as latency. While some of these are omittable, some may post a serious issue in big data processing environment, especially when the various dimensions of big data is high. Hadoop as a file distributed system which capable of parallel computing must overcome all characteristics of big data before the big data analyst could use it with certainty and confidence.

While many articles (both academic and non-academic) have reviewed MapReduce and Apache Pig's capabilities and limitations, most of the article's reviews are on top-down perspectives reviewing MapReduce and Apache Pig as a whole resulting in general commentary such as the review provided by DataFlair Team in [10].

A review that is coming from the bottom-up perspective is needed to serve as a guide for the new-comers to this parallel computing paradigm.

This paper intended to provide a practical review of Hadoop's MapReduce and Apache Pig capabilities and limitations from a down-to-earth perspective. As a result, two separate word counters were developed using both MapReduce (Python Stream) and Apache Pig Latin. The process is recorded, limitations and capabilities are provided from a coder and Hadoop user perspective.

II. RELATED WORD/ARTICLES

This section reviews a collection of articles (both online publication and academic journal/conference paper) and summarizes it into table format.

A. *MapReduce and its functions*

TABLE 1 Summary of MapReduce

Reference	Topic	Finding
[11]	Functions	A framework uses by Hadoop to executes functions on HDFS and aggregate results from clusters of machines
[12]	Language	Can be written in Java, Python, and C++

.

B. Apache Pig and its functions

TABLE 2 Summary of Apache Pig

Reference	Topic	Finding
[13]	Functions	Software for analyzing big data built on top of YARN and MapReduce
[13]	Language	Pig Latin, a high-level language similar to SQL
[14]	Extensibility	Users can create their own functions to perform special-purpose processing
[14]	Dependency	Relies on Hadoop HDFS, YARN
[14]	Framework	Uses a similar framework as MapReduce, but written in Pig Latin

C. MapReduce / Pig Limitations

As MapReduce and Pig are both utilizing the MapReduce framework, the following limitations can be applied to both MapReduce and Pig. The main difference between MapReduce and Apache pig would be in terms of the coding language as well as it's language capability.

TABLE 3 Limitations of MapReduce / Pig

Reference	Finding
[15]	Investigated MapReduce dependencies and GCC run-time on various systems (Linux, Image magick and Xen tools). MapReduce does not support multiple parallel mapper/reducer, instead, 1 large paralleled mapper has to be done, resulting in an inability to work effectively for a large class of applications.
[16]	Network bandwidth could be a scarce resource for file distribution and processing system like Hadoop
[17]	MapReduce has an issue with under heterogeneous environment. Hadoop HDFS and MapReduce assumed the datastore within the clusters are homogeneous. Despite data locality management can be applied, it affects MapReduce performance badly and requires many tedious works such as redesigning the dynamic distribution mechanism.

D. Research of word count (included text mining) using MapReduce or Apache Pig

TABLE 4 Word Count Related Literature

Reference	Topic and detail
[18]	Opinion Mining of Twitter Data using Hadoop and Apache Pig
[19]	Using MapReduce word count algorithm in research data analysis
[20]	Comparing MapReduce and FREERIDE for data-intensive applications. Finding suggested MapReduce is not suitable for a modest-sized dataset.
[21]	MapReduce performance is improved as data size is getting bigger.
[22]	Large Scale Data analysis using Apache Pig

III. METHODOLOGY

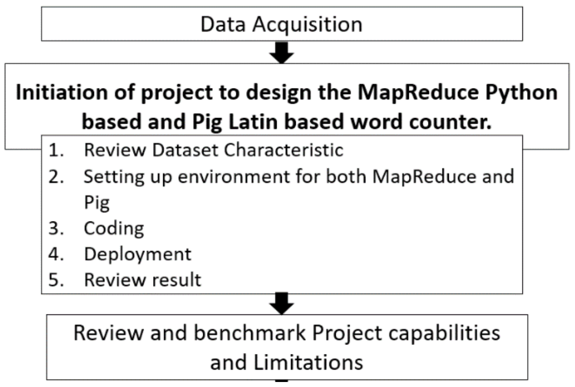

Data Acquisition
Initiation of project to design the MapReduce Python based and Pig Latin based word counter.
1. Review Dataset Characteristic
2. Setting up environment for both MapReduce and Pig
3. Coding
4. Deployment
5. Review result
Review and benchmark Project capabilities and Limitations

Fig. 1. Project process flow chart

A. Data Acquisition

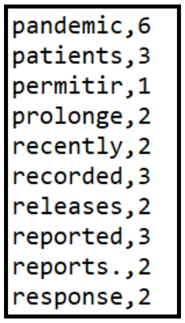
To review MapReduce and Apache Pig on its fullest level, a complex dataset that contains many forms of data (both structured and unstructured) is needed to test MapReduce and Pig capabilities in detail. To fulfilled this requirement a web scrape Python code was designed and implemented (Appendix I). As a result, a google search scraper scraped and recorded google search results by iterating the following keyword: Coronavirus, 'country', before:'date' after:'date'. The quote in the keywords refer to a list (e.g. list of 50+ countries, and list of dates range between 2020-4-12 to 2020-4-15) and will be iterate along with the search to generate a location and time-specific search result. The title and description of each search result are recorded following the date and country tag and lastly exported into a comma-delimited CSV file. Figure 2 refers to the first 2 rows of the dataset. The first 2 columns of the data are structured (namely date and country) and the third and fourth columns contain unstructured string data.

```
date,country,title,description
2020-04-13, Vietnam,Vietnam's total coronavirus cases climb
to 265 but no deaths | The ...,"5 days ago · HANOI:
Vietnam's health ministry reported five more confirmed
Covid-19 ( coronavirus) cases on Monday (April 13), taking
the country's tally to ..."
```

Fig. 2. First 2 Line of the Dataset with Column Name

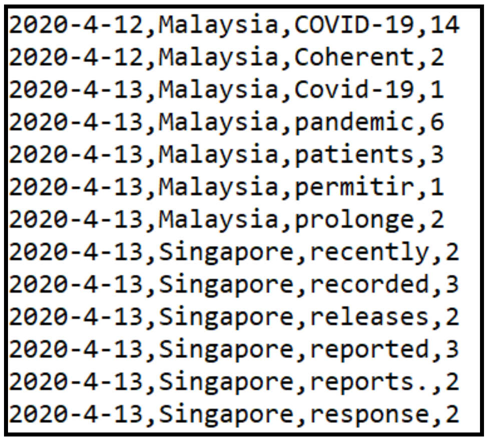
B. Word Counter Designation

Two separated word counter tool is designed, 1 with Python streamed MapReduce in mind while the other using Pig Latin SQL-like language. These tools have to be able to streamline and automate the process by inputting the dataset acquired from stage I and output a summarized word count as in figure 3. Moreover, improved from the simple word counter, the advanced word counter is aimed to be able to process the word by incorporating the concept of tagging, outputting the result that is date and country-specific as in figure 4.



```
pandemic,6
patients,3
permitir,1
prolonge,2
recently,2
recorded,3
releases,2
reported,3
reports.,2
response,2
```

Figure. 3. Expected Sample Output



```
2020-4-12,Malaysia,COVID-19,14
2020-4-12,Malaysia,Coherent,2
2020-4-13,Malaysia,Covid-19,1
2020-4-13,Malaysia,pandemic,6
2020-4-13,Malaysia,patients,3
2020-4-13,Malaysia,permitir,1
2020-4-13,Malaysia,prolonge,2
2020-4-13,Singapore,recently,2
2020-4-13,Singapore,recorded,3
2020-4-13,Singapore,releases,2
2020-4-13,Singapore,reported,3
2020-4-13,Singapore,reports.,2
2020-4-13,Singapore,response,2
```

Fig. 4. Expected Sample Output 2

C. *Review and Benchmark Project Capabilities and Limitations*

Alongside with the development, a continuous evaluation in terms of run-time, ease of development, the timeframe of development as well as coding difficulty is evaluated actively and summarized in section

IV. RESULT AND DISCUSSION

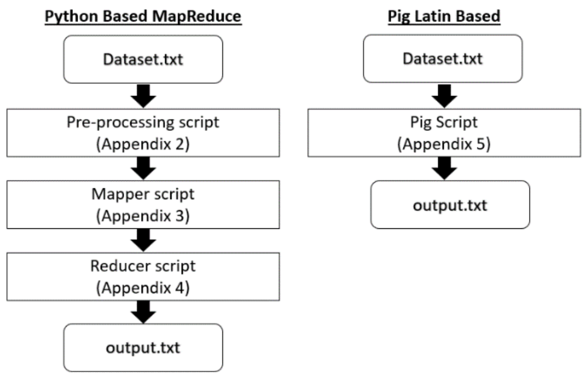


Figure 5: Data streamline and related appendix tag.

Two sets of word counter were developed to streamline the word counting process. Figure 5 summarized the developed streamline with its related appendix tag for the complete coding. The following section elaborates on the qualitative review that was generated throughout the process of the word counter development.

A. *Coding*

The coding aspect served as the main and most important differences among the other findings. As shown in figure 5, Python-based MapReduce requires a minimum of 3 independent files to carry out a complete MapReduce Process. Despite the uses of MrJob packages that may allow the development of 1 MapReduce file, it was avoided to prevent the issue with version incompatibility. Moreover, not only that pig used significantly less amount of file, the sum of codes required between Python MapReduce and Pig is significant where Python MapReduce used 33 lines of code while Pig uses only 13 lines of code.
Furthermore, the coding for Python-based MapReduce contains a higher level of complexity when comparing with Pig coding, Python-based coding used packages such as sys and pandas and several looping features. On the other side, pig's coding reveals a sense of simplicity where only a few codes of data wrangling execution were needed to streamline the process.

B. *Development timeframe*

While the development timeframe heavily relies on coding simplicity, it is expected that the development timeframe of Pig was significantly lower than its python counterpart. While it took more than 4 days for the researcher to develop a functional Python-based word counter on MapReduce, the Pig Latin version only took less than 2 hours to achieve its functional state.

C. *Runtime*

Despite the significant difference in terms of the number of codes employed, both versions of the word count took about a similar time in the processing part. (Tez enabled Tez took slightly longer but only marginally), suggesting despite having to code on a lower level and higher complexity Python environment, Python-based MapReduce doesn't significantly improve the runtime.

D. *Environment Setup and Dependencies*

It is important to realize that HDP 2.5 could only take Python 2 coding, which has impacted the researcher coding in a minor way, However, updating python ignoring the HDP version could result in crashes. While Python packages such as sys and pandas both having their dependencies to Python version. This had created a complicated layer of dependencies between the Hadoop and Python and could significantly increase the amount of time spent in setting up the environments. On the other side, pig seems to be much easy, everything is ready for the user upon finishing the setup of HDP itself, no additional package was installed.

In terms of processing dependencies, Python-based MapReduce was hardcoded to rely on stdin. and stdout. to bridge the communication between input file, mapper, reducer and output file. This heavily restricted coding flexibility when it comes to using MapReduce. Most of the traditional manner of Python coding (e.g. read data from stdin, edit, export data back to stdout) will not work in this situation as the old stdin file cannot be removed from the python coding and inputting new stdin will result in data redundancy. On the other side, the Pig does not possess such an issue.

E. Language Expandability

Both Python-based MapReduce and Pig-Latin provides their feature in expanding the language. Python supports the uses of packages while Pig supports the user-defined function. However, as described earlier regarding the issues of stdin limitation, Python coding pattern has been heavily limited, resulting in low flexibility despite the enormous amount of Python packages available.

F. Limitations Associated with Advanced Word Counter Development

Either version of the word counter is able to improved to the advanced word counter with the information tag as described in figure 4. This is due to several unique issues in its version. For Python, it is mainly due to the stdin restriction as grouping and processing the semi-structured dataset requires packages such as pandas and NumPy, but both packages will not work well with stdin's standard input. On the Pig side, it is mainly due to the characteristics of the SQL-like language. Processing complicated data that requires heavy general-purpose programming capability (e.g. Python) will require a significant amount of time to work around, but the performance is almost curtained to be affected when attempting to achieve a similar output using a domain-specific Pig Latin language. Despite spending additional times on the development, there is yet any workaround encountered to allow the creation of the mentioned advanced word counter, suggesting parallel processing a complex dataset that contains both structured and unstructured data requires a significant amount of time in development, let alone the tedious limitation as mentioned earlier for both versions of the languages.

This result explains that parallel processing seems impossible if not very hard to accomplished without proper planning and coding expertise. No single and simple way was found within the timeframe of this research that is capable to process such a complex dataset in a homogenous cluster of machines environment. Although it is important to understand there is always a workaround for the issues providing there is a good amount of time spent, however, the method itself may be far too tedious and may not worth the amount of time spent in developing it. Lastly, the described limitation also suggests the uses of Python-based MapReduce may be inferior in terms of flexibility when comparing to coding MapReduce in its native language: Java.

V. Recommendation and Conclusion

This paper provided a qualitative review from a coder and Hadoop user perspective in using both Hadoop's MapReduce and Pig environment by developing a similar word counter tool. The uses of both Python-based MapReduce and Pig in simple processing are viable depending on the coder programming language proficiency. The result suggested using Pig may be far better due to its simple environment setup, a low number of codes count and simple coding logic. Furthermore, there is no significant difference in terms of the runtime was encountered. However, both versions of the word counter failed to implement the information tag feature as described in advanced word counter due to their respective limitation. Despite the availability of using Python language in many aspects within the Hadoop ecosystem, its limitation must not be disregarded. The future researcher should focus on exploring MapReduce using Java-based coding to understand MapReduce capability on its fullest potential in processing complex datasets. Big data analysts who are attempting to process data with a great level of complexity will need to plan properly and acquire related expertise before able to utilize Hadoop's parallel file distribution and computing framework.

References

[1] M. N. Akhtar, J. M. Saleh, and C. Grelck, "Parallel processing of image segmentation data using Hadoop," *Int. J. Integr. Eng.*, vol. 10, no. 1, pp. 74–84, 2018, doi: 10.30880/ijie.xx.xx.xxxx.xx.xxxx.

[2] Y. Zhang *et al.*, "Parallel Processing Systems for Big Data: A Survey," *Proc. IEEE*, vol. 104, no. 11, pp. 2114–2136, 2016, doi: 10.1109/JPROC.2016.2591592.

[3] White T, *Hadoop: The Definitive Guide*. Sebastopol: O'Reilly Media, 2009.

[4] Venner J, *Pro Hadoop*. New York: A Press, 2009.

[5] C Lam and Warren J, *Hadoop in Action*. 2010.

[6] Hadoop, "Apache Software Foundation project home page." .

[7] K. Dwivedi and S. K. Dubey, "Analytical review on Hadoop Distributed file system," *Proc. 5th Int. Conf. Conflu. 2014 Next Gener. Inf. Technol. Summit*, pp. 174–181, 2014, doi: 10.1109/CONFLUENCE.2014.6949336.

[8] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," *3rd Nirma Univ. Int. Conf. Eng. NUiCONE 2012*, pp. 6–8, 2012, doi: 10.1109/NUICONE.2012.6493198.

[9] J. Dittrich and J. A. Quiané-Ruiz, "Efficient big data processing in Hadoop MapReduce," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2014–2015, 2012, doi: 10.14778/2367502.2367562.

[10] DataFlair Team, "13 Big Limitations of Hadoop & Solution To Hadoop Drawbacks," 2019. https://data-flair.training/blogs/13-limitations-of-hadoop/.

[11] Y. Tao, W. Lin, and X. Xiao, "Minimal MapReduce algorithms," *Proc. ACM SIGMOD Int. Conf. Manag. Data*, pp. 529–540, 2013, doi: 10.1145/2463676.2463719.

[12] IBM, "What is MapReduce?" https://www.ibm.com/analytics/hadoop/mapreduce.

[13] T. Ronald C, "An overview of the Hadoop/MapReduce/HBase Framework and its current Applications in Bioinformatics," *Nat. Methods*, vol. 7, no. 7, pp. 495–499, 2010, doi: 10.1038/nmeth0710-495.

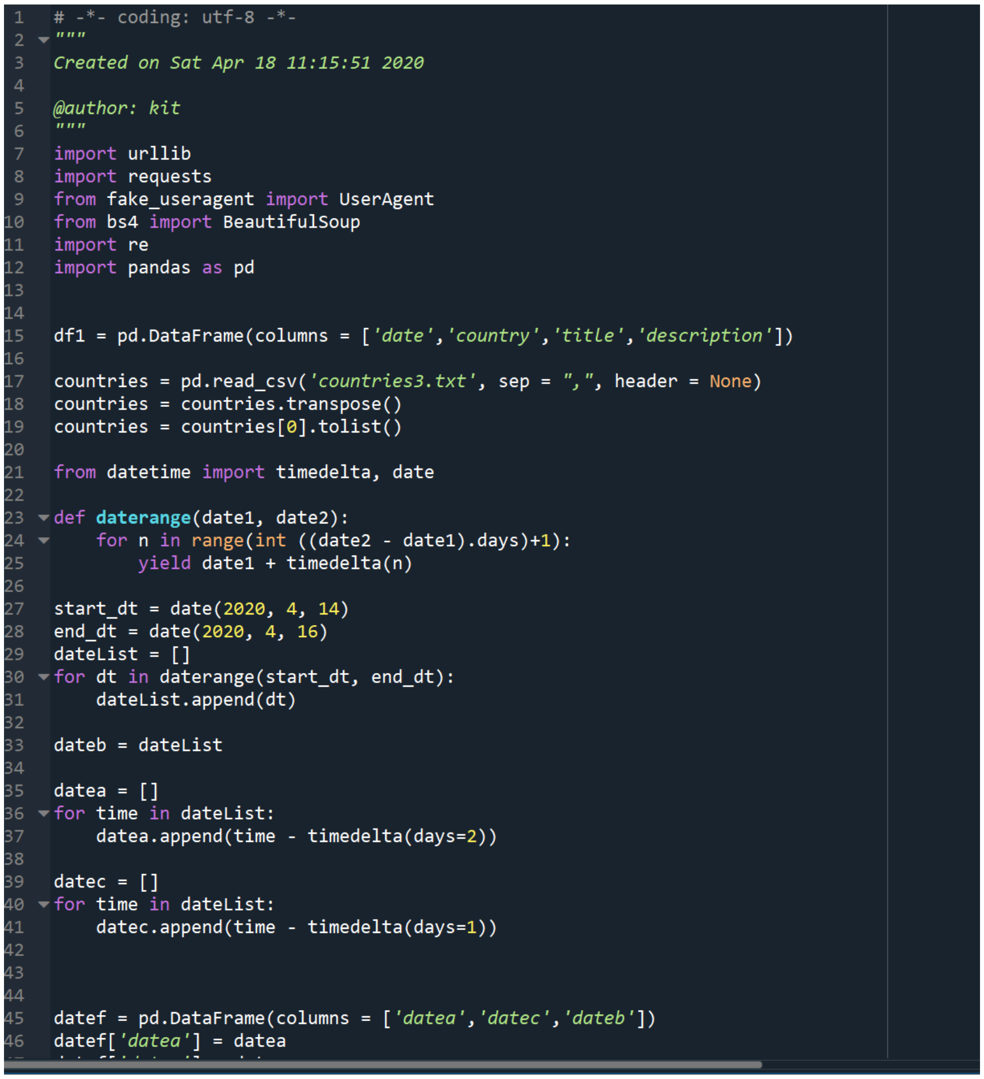
[14] P. R. A. Preethi and P. J. Elavarasi, "Big Data Analytics Using Hadoop Tools – Apache Hive Vs Apache Pig," vol. 24, no. 3, pp. 16–20, 2017.

[15] Z. Ma and L. Gu, "The limitation of MapReduce: A probing case and a lightweight solution," *Cloud Comput. 2010*, 2010, [Online].

Available: http://www.thinkmind.org/index.php?view=article&articleid=cloud_computing_2010_3_20_50031.
[16] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.
[17] S. Khalil, S. A. Salem, S. Nassar, and E. M. Saad, “Mapreduce Performance in Heterogeneous Environments: A Review.”
[18] A. Barskar and A. Phulre, “Opinion Mining of Twitter Data using Hadoop and Apache Pig,” *Int. J. Comput. Appl.*, vol. 158, no. 9, pp. 1–6, 2017, doi: 10.5120/ijca2017912854.
[19] C. Engineering, M. Sahane, S. Sirsat, R. Khan, N. S. Mahal, and R. Baug, “Analysis of Research Data using MapReduce Word Count Algorithm,” vol. 4, no. 5, pp. 8–11, 2015, doi: 10.17148/IJARCCE.2015.4542.
[20] W. Jiang, V. T. Ravi, and G. Agrawal, “Comparing map-reduce and FREERIDE for data-intensive applications,” *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, no. January, 2009, doi: 10.1109/CLUSTR.2009.5289199.
[21] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, “Evaluating MapReduce on Virtual Machines,” no. 2007, pp. 519–528, [Online]. Available: http://portal.acm.org/citation.cfm?id=1695659.1695708.
[22] J. Mehine, “Institute of Computer Science Large Scale Data Analysis Using Apache Pig Master ’ s Thesis Advisor : Satish Srirama Co-advisor : Pelle Jakovits,” *Time*, no. May, 2011.

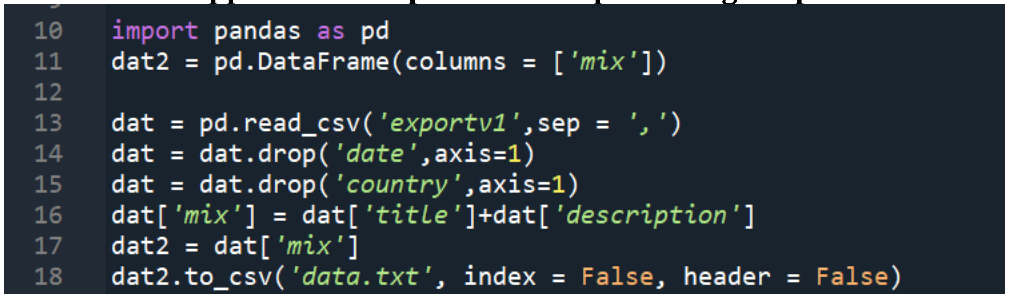
Appendices

Appendix 1: Google Search Scrapper (Please find the complete version in the link to GoogleDrive Resources)



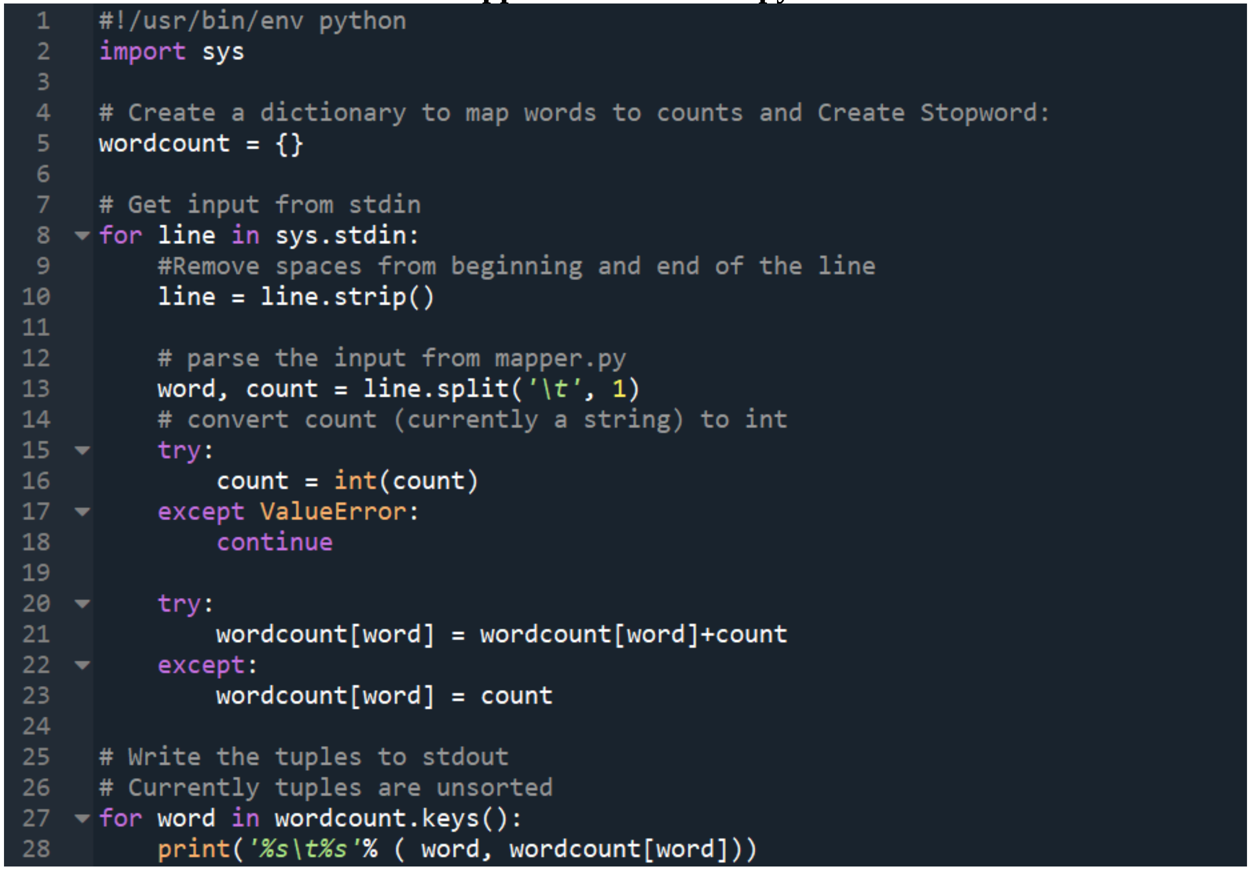
```python
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 18 11:15:51 2020

@author: kit
"""
import urllib
import requests
from fake_useragent import UserAgent
from bs4 import BeautifulSoup
import re
import pandas as pd


df1 = pd.DataFrame(columns = ['date','country','title','description'])

countries = pd.read_csv('countries3.txt', sep = ",", header = None)
countries = countries.transpose()
countries = countries[0].tolist()

from datetime import timedelta, date

def daterange(date1, date2):
    for n in range(int ((date2 - date1).days)+1):
        yield date1 + timedelta(n)

start_dt = date(2020, 4, 14)
end_dt = date(2020, 4, 16)
dateList = []
for dt in daterange(start_dt, end_dt):
    dateList.append(dt)

dateb = dateList

datea = []
for time in dateList:
    datea.append(time - timedelta(days=2))

datec = []
for time in dateList:
    datec.append(time - timedelta(days=1))



datef = pd.DataFrame(columns = ['datea','datec','dateb'])
datef['datea'] = datea
```

Appendix 2- MapReduce Pre-processing script



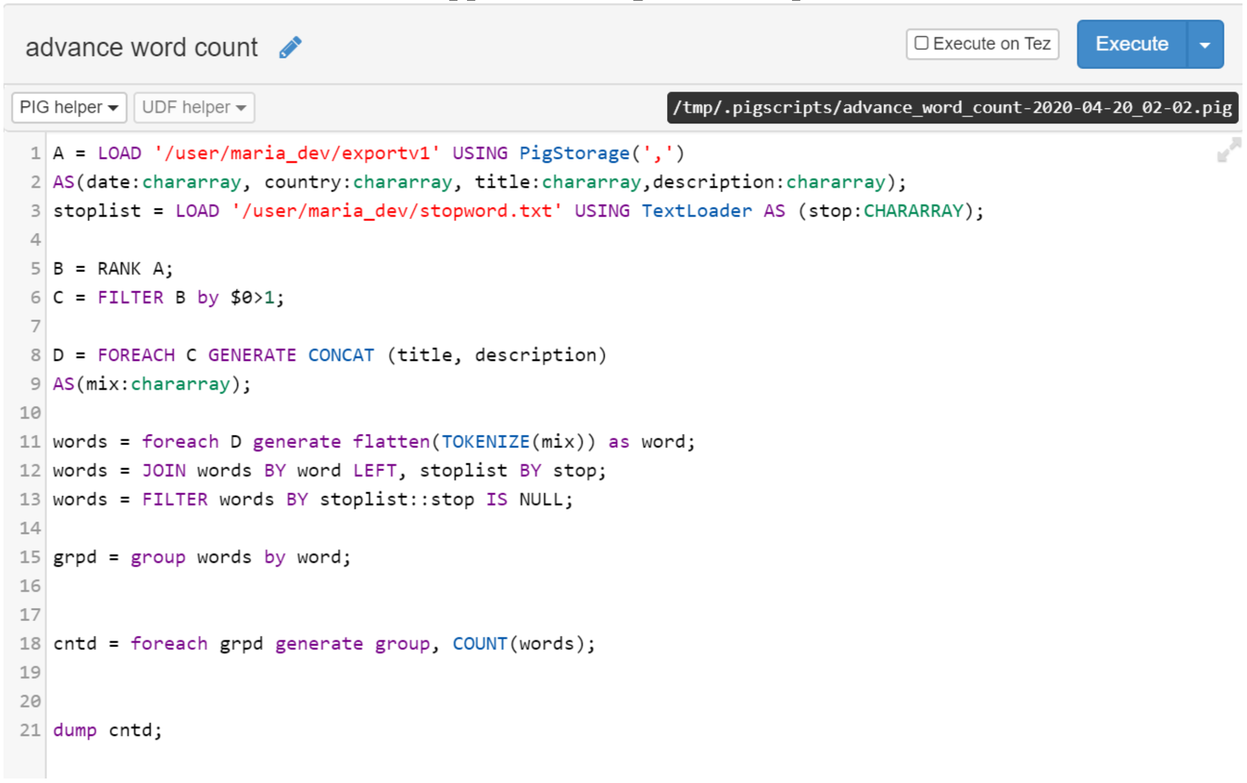
```python
import pandas as pd
dat2 = pd.DataFrame(columns = ['mix'])

dat = pd.read_csv('exportv1',sep = ',')
dat = dat.drop('date',axis=1)
dat = dat.drop('country',axis=1)
dat['mix'] = dat['title']+dat['description']
dat2 = dat['mix']
dat2.to_csv('data.txt', index = False, header = False)
```

Appendix 4- Reducer.py



```python
#!/usr/bin/env python
import sys

# Create a dictionary to map words to counts and Create Stopword:
wordcount = {}

# Get input from stdin
for line in sys.stdin:
    #Remove spaces from beginning and end of the line
    line = line.strip()

    # parse the input from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue

    try:
        wordcount[word] = wordcount[word]+count
    except:
        wordcount[word] = count

# Write the tuples to stdout
# Currently tuples are unsorted
for word in wordcount.keys():
    print('%s\t%s'% ( word, wordcount[word]))
```

Appendix 5- Pig Latin Script



```
A = LOAD '/user/maria_dev/exportv1' USING PigStorage(',')
AS(date:chararray, country:chararray, title:chararray,description:chararray);
stoplist = LOAD '/user/maria_dev/stopword.txt' USING TextLoader AS (stop:CHARARRAY);

B = RANK A;
C = FILTER B by $0>1;

D = FOREACH C GENERATE CONCAT (title, description)
AS(mix:chararray);

words = foreach D generate flatten(TOKENIZE(mix)) as word;
words = JOIN words BY word LEFT, stoplist BY stop;
words = FILTER words BY stoplist::stop IS NULL;

grpd = group words by word;


cntd = foreach grpd generate group, COUNT(words);


dump cntd;
```

Appendix 6: Link to complete resources included all coding info

Link:
https://drive.google.com/drive/folders/1zrt6jWZf0Qyj_7zqeAZucW6EFy-ua5lG?usp=sharing