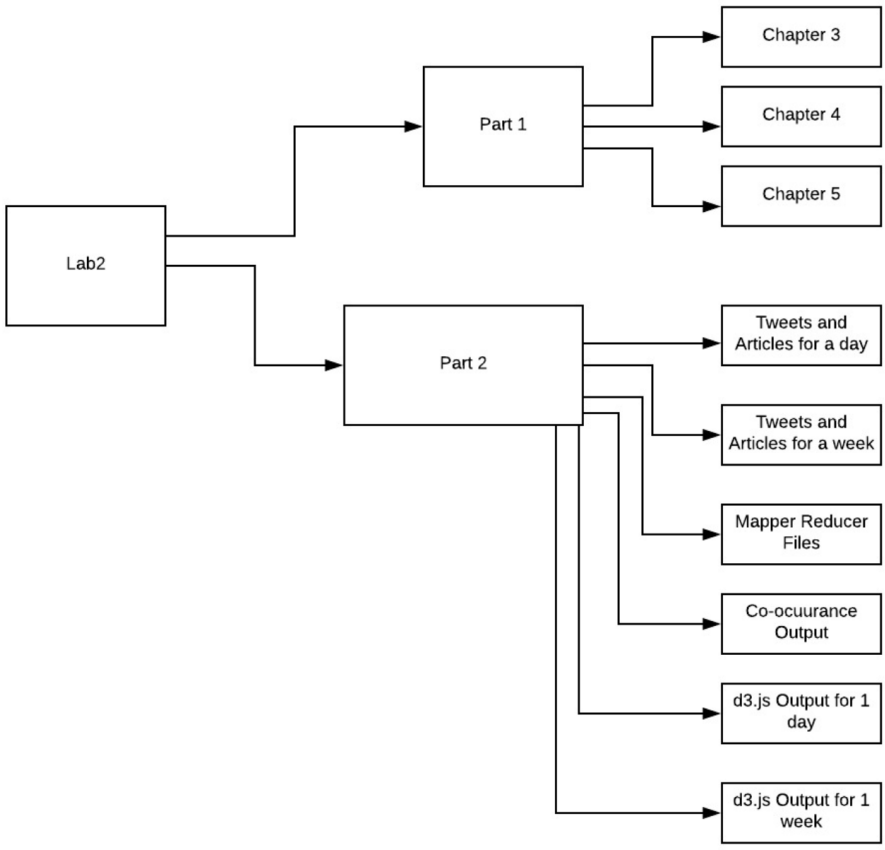
Data-intensive Computing

Lab 2: Data Aggregation, Big Data Analysis and Visualization: B. Ramamurthy

Done by:
Charanya Sudharsanan – 50245956 - csudhars
Prachi Shah – 50248748 – pshah

Directory Location:



Part1:
A number of basic python commands are tested and familiarized. The ipynb files are located in the above shown directory.
Directory location :
Chapter 3 – Lab2 - > part1 -> chapter3
Chapter 4 - Lab2 - > part1 -> chapter4
Chapter 5 - Lab2 - > part1 -> chapter5

Part2:
Part 2.A.

Topic: Trump

Collecting data from Twitter

- Tweets from Twitter API are collected using the keyword Trump after filtering out the retweets using twitteR. Done for one day as well as a week.
- Extracted 'text' column from the tweets data frame.
- Tweets are cleaned to process into mapper. (Cases , special characters, spaces, one letter words, redundant words are removed)
- Put each of the tweets into a newline to process the mapper for co-occurring words.
- Transfer tweets into a text file.

Collecting data from NY Times:

- Articles from NY Times API are collected using key word Trump for one day and later for one week.
- Removed url's containing videos and scraped only the content of the articles
- Tweets are cleaned to process into mapper. (Cases, special characters, spaces, one letter words, redundant words are removed)
- Put each of the articles into a newline to process the mapper for co-occurring words.
- Transfer articles into a text file.

Output of Step1:
Directory location :

- **Lab2\Part2\Tweets and articles for a day\NYTimes Articles Oneday\final_articles_newline_oneday.txt**

- **Lab2\Part2\Tweets and articles for a day\Tweets Oneday\tweets_newline_mapper_input_1dayreplaced.txt**

- **\Lab2\Part2\Tweets and articles for a week\TWITTER\ tweets_newline_mapper_input.txt**

- **Lab2\Part2\Tweets and articles for a week\NYTIME\ final_articles_newline.txt**

We now have the following data files.
One day's data:

- final_articles_newline_oneday.txt – One day's articles
- tweets_newline_mapper_input_1dayreplaced.txt – One day's tweets

One Week's data:

- final_articles_newline .txt – One Weeks articles
- tweets_newline_mapper_input_replaced.txt – One Week's tweets

Part 2.B.

Steps Involved:

1. We first Hadoop installed by using Oracle Virtual Box.
2. Basic commands are tested for the sample data available.
3. The following code snippet
    a. Starts Hadoop:
        i. start-hadoop.sh
    b. Creates a directory in the dhfs:
        i. hdfs dfs –mkdir user/hadoop/input
    c. Moves folder from Hadoop-home directory to hdfs directory:
        i. hdfs dfs -put /home/hadoop/books /user/hadoop/input
    d. Run Word Count:
        i. HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce- examples-2.6.4.jar wordcount input output
    e. View Output files:
        i. hdfs dfs -cat output/*
    f. Stop Hadoop:
        i. stop-hadoop.sh

Output File for Sample data

sample_output.txt

PART2.C:

Load the data aggregated in step (a) into the VM, two directories

- Start Hadoop:
    - start-hadoop.sh
- Create two folders called "twitterData" and "newsData"
    - hdfs dfs -mkdir /user/hadoop/input

- Push input files from local directory to hdfs(into the created input_file):

    Moving One day's tweet:
    - hdfs dfs -put /home/hadoop/lab2/ tweets_newline_mapper_input_1dayreplaced.txt /user/hadoop/input

    Moving One week's tweet:
    - hdfs dfs -put /home/hadoop/lab2/ tweets_newline_mapper_input_replaced.txt /user/hadoop/input

Moving One day's articles:

- hdfs dfs -put /home/hadoop/lab2/ final_articles_newline_oneday.txt /user/hadoop/input

Moving One week's articles:

- hdfs dfs -put /home/hadoop/lab2/final_articles_newline .txt /user/hadoop/input

PART 2.D

Directory Structure - > \Lab2\Part2\Mapper Reducer files\mapper.py

- Our Mapper and Reducer programs are written in python.
- It parses each of the datasets and into words.
- It removes stop words using nltk. We also removed words of size less than 4 as they are mostly verbs and pronouns.

Mapper Program:

mapper.py

```
#!/Users/prachishah/anaconda3/bin/python
"""mapper.py"""

import sys
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    #NLTK package helps to remove common englsh words
    stop_words = set(stopwords.words('english'))
    # increase counters
    for word in words:
        if not word in stop_words:
            #this condition is used to remove other common
            #words which are not removed by NLTK
            if len(word) > 4:
                #this is the output that Mapper emits to Reducer
                print (word, ' ', 1)
```

Reducer Program :

Directory Structure - > \Lab2\Part2\Mapper Reducer files\reducer.py

reducer.py

```
#!/Users/prachishah/anaconda3/bin/python

from operator import itemgetter
import sys

current_word = None
current_count = 0

#this dictionary stores all the words and their frequencies
#it will later be used to extarct top words based on application(Word
cloud or Co-occurence)
topWords = {}
#this dictionary stores the frequency of words in ascending order
sort = {}

#input is taken from stdin
for line in sys.stdin:
    line = line.strip()

    #splits the mapper input
    word, count = line.split(' ',1)
    try:
        count = int(count)
    except ValueError:
        continue

    #this sums the mapper input to calculate the WordCount
    if current_word == word:
        current_count += count
    else:
        if current_word:
            #this print stmt given the final wordCount frequency in part-
0000(R-1) file
            #print ('%s %s' % (current_word, current_count))
            topWords[current_word] = current_count
        #these stmts execute when new word comes as input
        current_count = count
        current_word = word

if current_word == word:
    #print ('%s %s' % (current_word, current_count))
    topWords[current_word] = current_count

import operator
length = len(topWords) - 1
#this sorts all the words in ascending order
sortedWords = sorted(topWords.items(), key=operator.itemgetter(1))

'''
```

```
#these stmt prepare intermediate files for producing co-occurence
application
with open('/Users/prachishah/Desktop/WordCount/scripts/temp.txt', 'w',
newline = '') as f:
    for i in range(10):
        #print(sortedWords[length - i][0]+'\t'+str(sortedWords[length -
i][1]))
        #this stmt writes top 10 words
        f.write(sortedWords[length - i][0])
        if i < 9:
            f.write(' ')

#these lines emit output for normal WordCount Problem
for x,y in topWords.items():
    print(x+'\t'+str(y))

'''

#the following stmts are used to produce output for WordCloud application
for i in range(100):
    sort[sortedWords[length - i][0]] = sortedWords[length -i][1]

import csv

with
open('/Users/prachishah/Desktop/WordCount/output/cloudArticlesDay.csv',
'w', newline = '') as f:
    w = csv.writer(f)
    w.writerow(['text','size'])
    for key, value in sort.items():
       w.writerow([key, value])
```

PART 2.E

D3.js Visualization:

Here, we create a web page with a word cloud to visualize the intensity of each word.

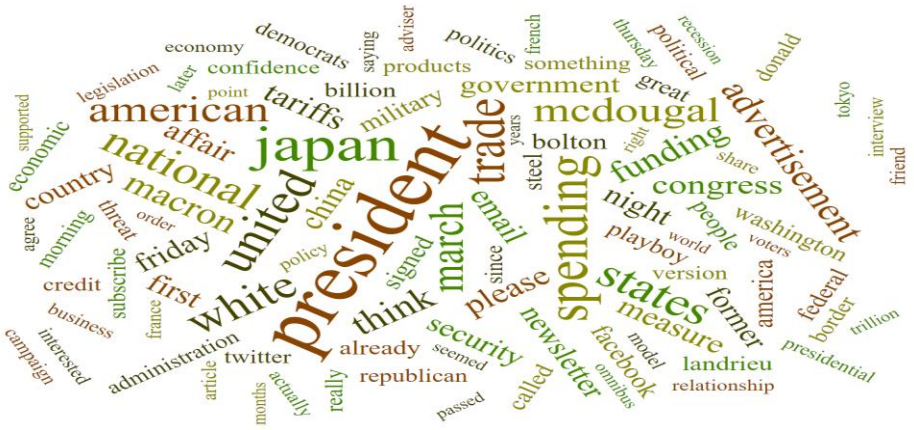
D3.js Output for One day:

1. **Output for tweets for a day:**
   Directory location : lab2 -> part2 -> d3.js output for 1 day -> example -> tweets_oneday.html

Word Cloud

2. **Output for articles for a day:**
   Directory location : lab2 -> part2 -> d3.js output for 1 day -> example -> articles_oneday.html

Word Cloud



PART 2.F.

D3.js Visualization:

Here, we create a web page with a word cloud to visualize the intensity of each word.(from the data collected for a week)

D3.js Output for One week:

1. **Output for tweets for a week:**
   Directory location : lab2 -> part2 -> d3.js output for 1 week -> example -> tweets_oneweek.html

Word Cloud

2. **Output for articles for a week:**

Directory location: lab2 -> part2 -> d3.js output for 1 week -> example -> articles_oneweek.html

PART 2.H.

Co-occurrence for top ten words:
Directory Structure - > \Lab2\Part2\Mapper Reducer files\mapper1.py

Mapper code to get the co-occurrences of top ten words :

```
#!/Users/prachishah/anaconda3/bin/python

import sys
import io


file = open("temp.txt",'r')
topwords = file.read().split(' ')
top = set(topwords)

for line in sys.stdin:
        line = line.strip()
        words = line.split()
        #'sets' has all the unique words in a tweet/article
        sets = set(words)
        #this statament checks if any of the top words exists in the
current tweet/article
        common = list(sets.intersection(top))

        #follwing statements emit pair of the following form
        #<one of the top 10 word, co-occuring word with the top 10 words>
        if common:
```

```
for x in common:
        for y in sets:
                if( x != y):
                        if(len(y) > 4):
                                print(x+' '+y)
```

Steps:

1. After finding the top ten words, we save them as temp.txt.
2. We read the file to find if every tweet or article has the word as in the temp.txt file.
3. In that case we find all the co-occurring word for each of the top ten words within its context (tweets for twitter data and articles for NY times article data.)

Reducer Program to process the co-occurring words:
Directory Structure - > \Lab2\Part2\Mapper Reducer files\reducer2.py

```
#!/Users/prachishah/anaconda3/bin/python

from operator import itemgetter
import sys

current_word = None   #one of the top 10 words
current_list = []     #this list is used to store co-occuring words with
current_Word

for line in sys.stdin:
        line = line.strip()
        word, coword = line.split(' ')


        if current_word == word and len(coword) > 4:
                if coword not in current_list:
                        current_list.append(coword)
        else:
                if current_word:
                        print(current_word,current_list)
                current_list = []
                current_list.append(coword)
                current_word = word

if current_word == word:
        #this outputs the final co-occuring words list for the top 10 words
        print(current_word,list(set(current_list)))
        sys.stdout.flush()
```

Output :

part-00000-articlesCoOneDay.txt

Other Outputs for tweets(one day and one week) and articles (One week)are in the directory – Lab2 ->Part2 ->Coccurance output

Comclusion

Hence, we have made a MapReduce application for WordCound, generating WordCloud and finding Co-occuring words.