MAP REDUCE

Harisaipravin SV - 17pw13

Venkata Niteesh - 17pw11

MapReduce is a processing technique and a program model for distributed computing.

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.

Map reduce Setup

```
import sys
import scipy
import numpy
import json
import bokeh
import tensorflow
import keras
import sklearn
import matplotlib
import pandas as pd
import seaborn as sns
```

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
!tar xf spark-3.0.0-bin-hadoop3.2.tgz

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"
```

```
pip install findspark
```

```
Collecting findspark
  Downloading https://files.pythonhosted.org/packages/fc/2d/2e39f9a023479ea798eed4351cd66f163
Installing collected packages: findspark
Successfully installed findspark-1.4.2
```

```
import findspark
findspark.init()
```

```
import pyspark
sp = pyspark.SparkContext()
print(sp)
```

```
<SparkContext master=local[*] appName=pyspark-shell>
```

The setup involves setting up the os variable and changing the default and safe modes, Here findspark init does most of the variable setting job and we just use the pyspark later on.

Map reduce Intro

```
namesList = ['hari','sai','pravin','grandhi', 'venkata','sai', 'swathi', 'yamini']
names = sp.parallelize(namesList, 4)

print (names)
names.collect()
```

```
ParallelCollectionRDD[16] at readRDDFromFile at PythonRDD.scala:262
['hari', 'sai', 'pravin', 'grandhi', 'venkata', 'sai', 'swathi', 'yamini']
```

Here we initialize a list of our classmate names, to give a try on map reduce functions , thereby attaining a in depth knowledge on it.

Further, we parallelize the list to the proper RDD container which could be understood by map reduce.

map reduce is highly flexible, we developed a function to demonstrate it.

```
def rollno(word):
    return ('17pwss - ' +word)
```

On trying the function for our dataset, it can be seen that all of our data are added with 17pss in front of their names. This can be used to group classes.

```
rollnumber = names.map(rollno)
print (rollnumber.collect())
```

```
['17pwss - hari', '17pwss - sai', '17pwss - pravin', '17pwss - grandhi', '17pwss - venkata', '17pwss - sai', '17pwss - swathi', '17pwss - yamini']
```

MAP FUNCTION :-

Map function that processes a key/value pair to generate a set of intermediate key/value pairs

```
pairkeynames = names.map(lambda na: (na, 1))
print (pairkeynames.collect())
```

```
[('hari', 1), ('sai', 1), ('pravin', 1), ('grandhi', 1), ('venkata', 1), ('sai', 1), ('swathi', 1), ('yamini', 1)]
```

Here .map is used to map every element in our list to unit value (1).

REDUCE FUNCTION :-

```
uniquenames = (names
                        .map(lambda na: (na, 1))
                        .reduceByKey(lambda o,p: o+p)
                        .collect())
print (uniquenames)
```

```
[('hari', 1), ('sai', 2), ('grandhi', 1), ('venkata', 1), ('yamini', 1), ('pravin', 1), ('swathi', 1)]
```

We use .reduceByKey to call the reduce function, and we specify the operation that needs to be done there.

Finally we could get the map reduced result which has clubbed the similar names together.

Now that we know how it works, let's implement it for a big dataset.

DATASET & PROBLEM STATEMENT :-

“My bounty is as boundless as the sea”

The dataset contains the dialogues from the famous **romeo juliet** drama, we’ll try to analyse the speech and see which words are repeated the most number of times.
Other dataset would be common **stop words** as we should ignore them.

DATA PREPROCESSING

```
com_list1 = pd.read_csv('com_words.txt')
com_list1 = list(com_list1['stop_words'])
print(com_list1)
```

```
['a', "a's", 'able', 'about', 'above', 'according', 'accordingly', 'across',
```

```
com_list2 = list(map(lambda element: element.capitalize(), com_list1))
print(com_list2)
```

```
['A', "A's", 'Able', 'About', 'Above', 'According', 'Accordingly', 'Across',
```

```
com_list3 = list(map(lambda element: element.upper(), com_list2))
print(com_list3)
```

```
['A', "A'S", 'ABLE', 'ABOUT', 'ABOVE', 'ACCORDING', 'ACCORDINGLY', 'ACROSS',
```

```
com_words = com_list1 + com_list2 + com_list3
```

Since our Speech dataset is not entirely in upper or lower case, we manipulate our stop words to small, upper and Camel cases for further processing.

LOADING DATASET

```
file_words = sp.textFile('romeojuliet.txt')
speech_words = file_words.flatMap(lambda x: x.split())
print('File has total of {} words.'.format(file_words.count()))
```

```
File has total of 4420 words.
```

```
print(speech_words.take(50))
```

```
['1595', 'THE', 'TRAGEDY', 'OF', 'ROMEO', 'AND', 'JULIET', 'by', 'William',
```

The dataset contains 4420 words and we are Mapping the words from the file using flatMap to further use them in Map reduce.

FILTERING

```
output_words = speech_words.filter(lambda x: x not in com_words)
print(output_words)
print(output_words.collect())
```

```
PythonRDD[110] at RDD at PythonRDD.scala:53
['1595', 'TRAGEDY', 'ROMEO', 'JULIET', 'William', 'Shakespeare', 'Dramatis',
```

Here the common words are removed from the speech words, and thus we have filtered our dataset to have only proper values.

FREQUENT WORDS USING MAP REDUCE

```
frequent_words = (output_words.map(lambda w: (w, 1)).reduceByKey(lambda x,y: x+y))
print(frequent_words.collect())
```

```
[('TRAGEDY', 1), ('Shakespeare', 1), ('Dramatis', 1), ('Personae', 1), ('Chorus.', 3), ('P
```

```
print(frequent_words.takeOrdered(10,key=(lambda x: -x[1])))
```

```
[('thou', 228), ('Rom.', 163), ('thy', 144), ('Jul.', 117), ('Nurse.', 100), ('thee', 89),
```

As discussed earlier, we have mapped and reduced all the elements together using our mapreduce function and now we can see that thou is the word that is repeated the highest number of times and there are 228 repetitions.

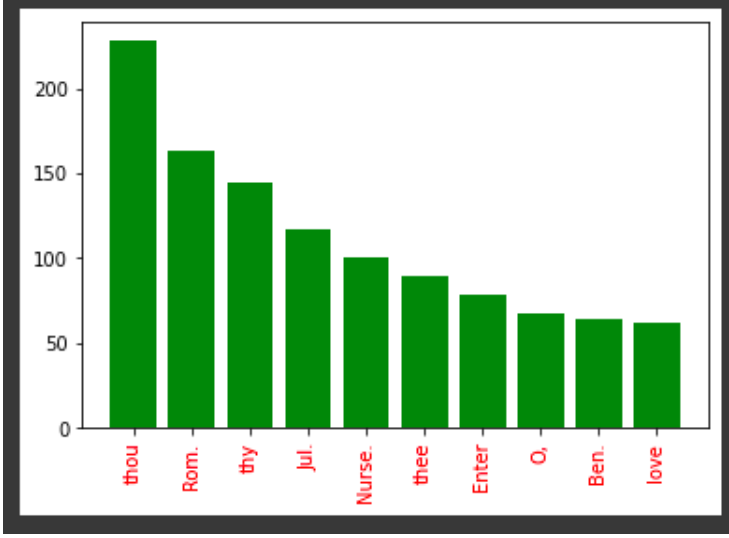
The .map maps all the words from the dataset to 1. Which is the default value.

Next,reducebyKEY inbuilt reducer groups the key value pair by their key value. Finally we have ordered them by descending order to get the max repeated element.

VISUALIZATION

```
plotter = frequent_words.takeOrdered(10, key=(lambda x: -x[1]))
plot_frame = pd.DataFrame(plotter)


plt.bar(list(range(1,11)), plot_frame[1], color = "green")
plt.xticks(range(1, 11), plot_frame[0], rotation='vertical', color = "red");
```



The final visualization gives the order of the frequently used words in the Rome-juliet play.