Problem Description:

The solution of the problem is divided into three parts and two lists of books are provided namely shortlist and longlist for carrying out the task.

Part 1: Write a code that calculates the occurrence of all words in the provided books on a per book basis in python using *pyspark*. Measure the execution time by varying the number of reducer and compare the result with the HW1 where same task was completed using *pydoop*.

Part 2: Write a *pyspark* code using mllib to calculate tf-idf (Term frequency, Inverse Document Frequency) for unigram and bigram for given list of books and measure the execution time by varying the number of reducer.

Part 3**:** Write a *pyspark* code to determine the feature vectors for words in the largbook-collection using the word2vecclass of Spark Mlib and measure the execution time by varying the number of reducer. In the next part, find out synonyms of the top 10 words and justify the methodologies used to select the words.

Output of these experiments are further analyzed using graphs.

Solution Strategy:

First read through the problem to understand the what is expected as a solution, use the with following mentioned skillset this task can be completed. Tools required are Map-Reduce framework, HDFS file system, Whale Cluster and python to write the required program to do the analysis.

Following are the skillset required to be able to complete the HW2.

1. Acquire basic knowledge of Unix commands such as ls, cd, mkdir, rm and SSH are required.
2. Next, understand how Map-Reduce framework works and basic commands to HDFS for the ease of working with it.
3. Coding skills in python, string parsing using regex.
4. Learn how to use spark library in python, feature hashing, vectorization of words.
5. Preparing graphs in MS excel.

Description of how to run your code

My codes are available in the in the *whale.cs.uh.edu* server under the following directory.
/home2/cosc6339/bigd27

- First login to *whale.cs.uh.edu* server using bigd27 credentials. Once you are in the home directory you will find the following python code using *ls* command.

```
bigd27@whale:~> ll Assig*py
-rw-r--r-- 1 bigd27 hadoop  643 Mar 30 18:18 Assignment2_Part1.py
-rw-r--r-- 1 bigd27 hadoop 1198 Mar 30 18:30 Assignment2_Part2_Bigram.py
-rw-r--r-- 1 bigd27 hadoop 1203 Mar 30 12:28 Assignment2_Part2_Unigram.py
-rw-r--r-- 1 bigd27 hadoop 3259 Mar 30 18:19 Assignment2_Part3_Synonyms.py
-rw-r--r-- 1 bigd27 hadoop  978 Mar 30 18:46 Assignment2_Part3_Word2Vec.py
```

- Below is the command to run a MapReduce job.

spark-submit --master yarn Assignment2_Part3_Word2Vec.py /cosc6339_s17/books-longlist --num-executors 2

Vary the number of reducer to measure the output with different number of reducer.

- Below is the command to fetch the output form HDFS and combine multiples output files into one

hdfs dfs -getmerge /bigd/output/part- / output.txt*

To run a MapReduce job on the cluster, the input data should be in HDFS, not in the local home directory, and the result in an output folder in HDFS.

Explanation of the Code:

Part1: Following is the explanation of functions used in the code.

sc = SparkContext(appName="xyz ")

In spark `SparkContext` object is automatically created in the shell for Python: accessible through the `sc` variable. With this command, we can set an app name as we desire.

wholeTextFiles:
This function reads each file as a single record and returns the file path as key and the content of each file as value.

```
def fileProcessor(x):
    fileName = x[0]
    fileName = re.sub('[\+]+', '', fileName).split("_")[-1]
    words = re.sub('[^A-Za-z]+', ' ', x[1].lower()).split()
    newWordList = [x + ' ' + fileName for x in words]
    return newWordList
```

This file processor function receives the file name and content as a key value pair get rid of the unwanted character using regex new list of words along with the filename.

saveAsTextFile("/bigd27/output")
This function saves data as a textfile in the specified location in HDFS.

Part2: Following is the explanation of functions used in the code.

sqlContext = SQLContext(sc)
dataFrame = sqlContext.createDataFrame(fileRDD, ["fileName", "content"])

One use of Spark SQL is to execute SQL queries written using either a basic SQL syntax. A Data Frame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python. Using these two command and RDD can be converted into dataframe.

ngram = NGram(n=1,inputCol="words", outputCol="bigrams")
unigramDataFrame = ngram.transform(filteredDataFrame)

Using these command dataframe is converted into Unigram or Bigram based on the value of n.

hashingTF = HashingTF(inputCol="bigrams", outputCol="rawFeatures", numFeatures=100000)
hasingData = hashingTF.transform(unigramDataFrame)
idf = IDF(inputCol="rawFeatures", outputCol="TF-IDF_Unigram")
idf_fit = idf.fit(hasingData)

Using the above functions, term frequency is calculated based the hash values of words. It also calculates the hash values of the words and finally calculates tf-idf.

Part3: Following is the explanation of functions used in the code.

The following function ultimately returns top ten list of words. First the RDD was parsed through fileParser followed by map and then reducedbyKey. Till this part the code calculated the number of occurrence of each word. After that the they were sorted based on number of occurrences and then top ten words were selected using take() function.

```
topTenElements = sc.textFile(sys.argv[1], use_unicode=False)\
                    .flatMap(lambda line: line.split())\
                    .filter(lambda x: fileParser(x))\
                    .map(lambda w: (w, 1))\
                    .reduceByKey(add)\
                    .map(lambda (x, y): (y, x))\
                    .sortByKey(False)\
                    .take(10)
```

This part of the find the synonyms of words from the toptenlist and calculates 5 synonyms for each word.

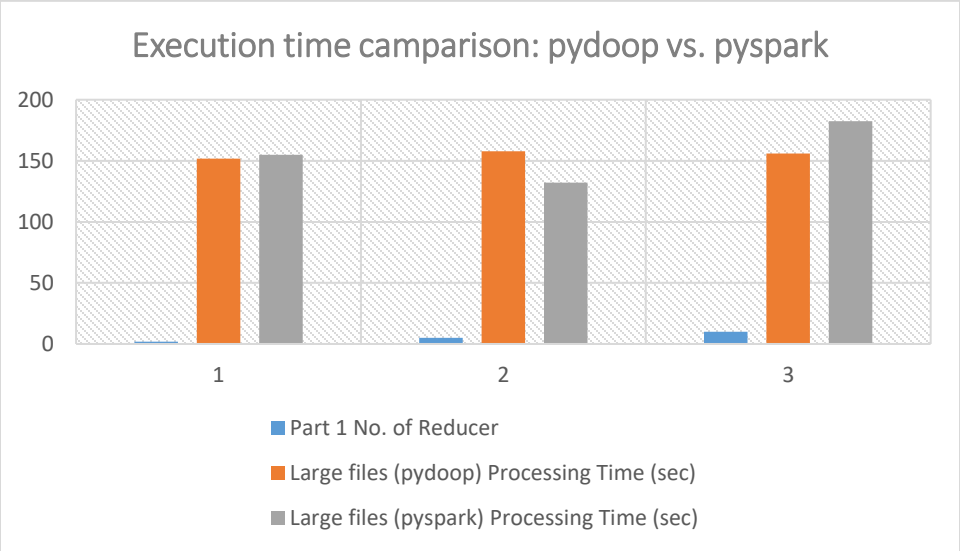
```
for word in finalList:
    FinalListOfSynonyms += 'Word: ' + "\n" + str(word).upper() + "\n" + 'Synonyms are' + "\n"
    synonyms = model.findSynonyms(word, 5)

    for synonym, cosine_distance in synonyms:
        FinalListOfSynonyms += str(("{}: {}".format(synonym, cosine_distance)) + "\n")
```

Results:

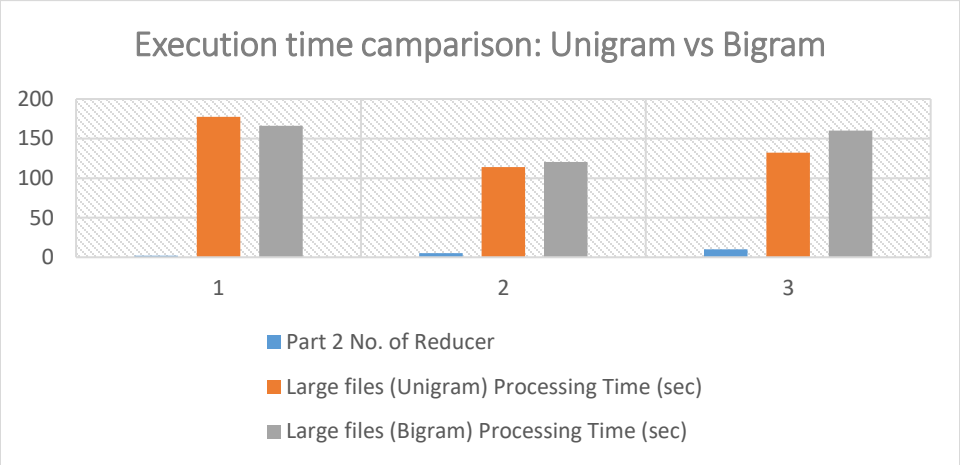
Part 1	Large files (pydoop)	Large files (pyspark)
No. of Reducer	Processing Time (sec)	Processing Time (sec)
2	151.8	154.8
5	157.8	132
10	156	182.4

From the above table, we could infer that for 5 number of reducer we get the best processing time and for 10 the performance degrades in case of pyspark. However, in case of pydoop performance degrade with the increasing number of reducer. If we compare pydoop and pyspark with the same number of reducer, then pydoop gives us better performance in two out of three cases.



Part 2	Large files (Unigram)	Large files (Bigram)
No. of Reducer	Processing Time (sec)	Processing Time (sec)
5	177.6	166.2
10	114	120.6
15	132	160.2

In Part 2, in both the cases we get the best processing time with 10 reducers. In case of Unigram and Bigram with 5 reducers we are getting the worst performance. However, in processing time comparison Unigram wins in two out of three cases.



Part 3: Long list (word2vec)	
No. of Reducer	Time taken (sec)
5	610
10	541
15	589

In part 3, we get the best processing time with 10 reducers and worst when only 5 reducers were used.