Big Data Analysis

Report by:- Harish Devathraj
PSID: 1800866

Question:

Part 1:

Write a pyspark code to determine the 1,000 most popular words in the document collection provided. Please ensure that your code removes any special symbols, converts everything to lower case (and if possible remove stop words, destemming, etc.).

Part 2:

a. Write a pyspark code to create an inverted index for the 1,000 words determined in Part 1. The inverted index is supposed to be of the form
term1:doc1:weight1_1,doc2:weight2_1,doc3:weight3_1,...term2:
doc1:weight1_2,doc2:weight2_2,doc3:weight3_2

b. Measure the execution time of the code for the large data set for 5, 10 and 15 executors.

Part 3:

a. Write a pyspark code to calculate the similarity matrix S with each entry of S being With V being the vocabulary (determined in part 1) and the weights having been determined in part 2.

Part 4:

Provide a list of the 10 most similar (or identical) pair of documents from the large data set.

Problem Description:

Part 1

For this part of the problem, a directory in the HDFS should be read. The directory contains multiple big Txt files.

Our goal is to:

1) Remove Special characters.
2) Extract top 1000 frequently used words in whole data set.

Part 2

In this part, we have to find the inverted index of the top 1000 words which were extracted previously.

Which is of the form as shown below:

Term1: doc1: Weight1_1, doc2: weight2_2….
Where, term1: 1st word
Weight = Count of that word in that particular file.

b. We then compare the execution times with 5, 10 and 15 executors.

Part 3

In this part, we have to determine the similarity matrix S with each entry with V being the vocabulary and the weights

S(docx, docy) = E(weight_docx * weight_docy)

b. We then compare the execution times with 5, 10 and 15 executors.

Part 4

Need to identity 10 identical pair of documents from large dataset

Solution Strategy:

Part 1

Firstly, we need to load the data-set. For this we first read the file as a RDD.
a) We split the contents in the file by spaces and lowercase all the content.
b) Remove the special characters and numbers using regular expression.
c) We apply map method and give a word count 1 to all the words in all the text files.
d) ReduceByKey takes each word as key and it adds up all the words and displays the count for each word.
e) Using takeOrdered in built function in pyspark we have mentioned 1000 to display 1000 words.
f) Finally we display the top 1000 frequently used words in the entire dataset.

Part 2

In this part we are finding the inverted index
a) We use sc.wholeTextFile('/path') to read the data set. We wholeTextFile returns us the filename and the contents of that file (name, contents).
b) We process the contents into a function linesTowords to split the text file, lower all the contents and then remove the special characters.

We Map every word to with a filename and count of the word and store them onto a list.

```
.map(lambda ((word, name), count): (word, [name, count]))
```

This gives us the result in format word : [(filename : weight), (filename : weight)]
Then this result is saved to a text file using saveAsTextFile() function.

Part 3

1)We use the RDD obtained from part 2
2) This list is created for every word in the Vocabulary

3) We create a which looks this type list[] = [('d1', 4), ('d2', 8), ('d3', 0)…]

4) We have mapped each filename to a sequence number
Such as filename1.txt 1, filename2.txt 2, filename3.txt 3, …… filenameN.txt N

5) This code below finds the similarity matrix

```
doc1 = doc_map[list[i][0]]
doc2 = doc_map[list[j][0]]
if doc1 < doc2:
    similarity[doc1][doc2] += (list[i][1] * list[j][1])
else:
    similarity[doc2][doc1] += (list[i][1] * list[j][1])
```

6) Now similarity[1][2] gives us the similarity between document 1 and 2
Similarity[5][78] gives us the similarity between 5 and 78

Part 4

We can scan through the 2D matrix and find out the top 10 elements in the matrix Whose index gives the us the most similar documents.

Running the code:

Part 1

To run this part of the code place ‘part1.py’ in your home directory. Use the following command to run:

spark-submit --master yarn --num-executors 10 part1.py /cosc6339_hw2/gutenberg-500/ /bigd30/hw2out1

To get this output in your home directory use:

hdfs dfs -get /bigd30/ hw2out1 myout1

Now, you should have a directory ***‘myout1’*** in your home directory. Note that this step may take some time(depends on internet speed) as the files created are large. The files are present in my home directory and the hdfs directory with the same names.

Part 2

To run this part

spark-submit --master yarn --num-executors 10 part2.py /cosc6339_hw2/gutenberg-500/ /bigd30/hw2out2

spark-submit --master yarn --num-executors 5 part2.py /cosc6339_hw2/gutenberg-500/ /bigd30/hw2out3

spark-submit --master yarn --num-executors 15 part2.py /cosc6339_hw2/gutenberg-500/ /bigd30/hw2out4

To get this output in a single merged file in your home directory use the following commands:

hdfs dfs -getmerge /bigd30/hw2out2/part-* .txt out

Part 3

To run this part

spark-submit --master yarn --num-executors 10 part3.py /cosc6339_hw2/gutenberg-500/ /bigd30/hw2out6

spark-submit --master yarn --num-executors 5 part3.py /cosc6339_hw2/gutenberg-500/ /bigd30/hw2out7

To get this output in a single merged file in your home directory use the following commands:

```
hdfs dfs -getmerge /bigd30/hw2out2/part-* .txt out1
```

Results:

Resources used:

The Hadoop cluster used additionally is whale. It has been allotted a memory of 540 GB across 720 VCores. The cluster currently has 44 active nodes.

50 Appro 1522H nodes (whale-001 to whale-057), each node with

- two 2.2 GHz quad-core AMD Opteron processor (8 cores total)
- 16 GB main memory
- Gigabit Ehternet
- 4xDDR InfiniBand HCAs (not used at the moment)

Network Interconnect

- 144 port 4xInfiniBand DDR Voltaire Grid Director ISR 2012 switch (donation from TOTAL)
- two 48 port HP GE switch

Storage

- 4 TB NFS /home file system (shared with crill)
- ~7 TB HDFS file system (using triple replication)

Further for each step the following resources are needed-

This Project uses Pyspark SparkConf which is the configuration for a Spark application. It is used to set various Spark parameters as key-value pairs. SparkContext is the main entry point for Spark functionality. A SparkContext represents the connection to a Spark cluster, and can be used to create RDDs and broadcast variables on that cluster.

Measurements Performed:

Part 1

The following measurements were performed on the medium dataset (Gutenberg-500):

1) Time required to run part1.py

Part 2

The following measurements were performed:

1) Time required to run with 5 executors
2) Time required to run with 10 executors
3) Time required to run with 15 executors

Part 3

The following measurements were performed:

4) Time required to run with 5 executors
5) Time required to run with 10 executors

Part 4

6) Time required running part4.py

Findings:

The following were the findings:

Note that these runs were tested when there was a very little or no load on the cluster. It had either 0 or at most 1 application running simultaneously on the large dataset.

PART 1

This part was executed **3 times**. The following were their execution times:

1. 7mins, 53 sec
2. 5mins, 57 sec
3. 6mins, 16 sec

On an average, we can say that finding top 1000 frequently words in a medium dataset takes 6mins, 30 seconds.

Note: As expected on all the 3 runs the size of output files were same.

PART 2

Note that these runs were tested when there was a medium load on the cluster. It had either 4 or more than 5 (up to 10) applications running simultaneously and was performed using medium dataset.

(BEST TIMING OUT OF 3 RUNS)

Number of Executors*	Time Elapsed
5	27mins, 39sec
10	24mins, 34sec
15	22mins, 34sec

Part 3

Note that these runs were tested when there was a medium load on the cluster. It had either 4 or more than 5 (up to 10) applications running simultaneously and was performed using medium dataset.

(BEST TIMING OUT OF 2 RUNS)

Number of Executors*	Time Elapsed
5	24mins, 3sec
10	22mins, 34sec

Final Comments:

The execution time depends on many factors. Such as
1) How efficient the code is
2) What are the resources available to you at that point of time.
3) Type of resources used.
4) Num of executers used and even user can specify the amount of memory and corer's to be used to submit a pyspark job.
5) The results varies between all the types.