**Assignment-2**

**Big Data Analytics**

**1637386**

**Chethana Dukkipati**

**Problem Description:**

Using Pyspark, finding out the multiple representations of the same real-world object for huge amount of data. For all the huge amount of data, finding out the duplicate objects takes a huge amount of time leads to quadratic complexity. Hence Pyspark is used to reduce this complex behavior.

**Solution Strategy:**

We are going to find out the frequency of words in the documents available. The top 1000 words among these are considered for obtaining the top duplicate documents by removing the stop words. The stop words from the “ntlk” library is read using python and are removed from the words available. Along with the stop words, the punctuation marks are also removed from the words. The punctuations are replaced with the space. For the 1000 words obtained, find out the weighted value for all the words. The weighted value is calculated for each document by considering the frequency of word in the document to the total number of words in the document. This procedure is repeated for every document. Once after the weights are obtained, the similarity between the two documents is found out by adding up the weighted value of each word among these two documents. The top 10 document pairs of these similarity matrix are considered as the duplicate documents.

The following description shows the implementation of above strategy using pyspark.

In the file name question21.py in the attached folder contains the implementation for the collection of 1000 words. The input file is read and is send to the function that splits the lines of the text file with spaces after converting into lower case. The data thus obtained will be given to map that adds up the frequency of words and take() will take the top 1000 words. The output first part is sent as input to the second part and are split to get the name of the word and frequency of word. The document files are read using the wholetextfile() and total number of words in the document are found out. The map is used to store the data as word and documents with the weight value of the word. This output is passed to third program which is passed to a function with 2 loops that will multiply the weights of two documents and store the results of them as per document pair along with similarity value. The take function is used to get the top 10 records which are duplicate documents along with similarity value.

**Description of Resources Used:**

To create and execute the application of mapreduce to process the huge amount of data provided, we used a cluster and server. The IP address of server we used to create the application is whale.cs.uh.edu and the execution of code will take place in cluster <http://whale.cs.uh.edu:8088/cluster>. There are different users available in the server and each are allocated with a username.

Few packages of python like pydoop and functions of the package like mapreduce are imported to implement the application. For user Interface, MobaXterm is been used and for testing the accuracy of the results obtained through mapreduce, the code is implemented in python through pycharm and results are compared for correctness and notepad++ for viewing the files.

**Cluster Information:**

The clusters consist of a login node (whale.cs.uh.edu) and several compute nodes. The cluster shares home directories crill but are otherwise separate. The only access method to whale from the outside world is by using ssh.

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 now)

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)

Different functions are used for performing the functioning like map, flatmap, operator package, reducebykey and groupby functions.

**Description of Measurements Performed:**

We have calculated the frequency of words, weighted values of these words and similarity values and then taken top 10 similar documents. The performance is measured by considering the execution times for different executors for all the parts.

For each part, we have taken the time of execution for 5,10 and 15 executors. Formulas used in part 2 and 3 are as follows:

term1: doc1:weight1\_1,doc2:weight2\_1,doc3:weight3\_1,… term2: doc1:weight1\_2,doc2:weight2\_2,doc3:weight3\_2,… …

where weightx\_y is: no. of occurrences of termx in document y / total number of words in document y

S(docx, docy) = 𝑡ϵ𝑉(𝑤𝑒𝑖𝑔ℎ𝑡\_𝑑𝑜𝑐𝑥× 𝑤𝑒𝑖𝑔ℎt\_𝑑𝑜𝑐𝑦)

**Results:**

The time of execution for different executors of different parts are observed as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Parts/Executors | 5 | 10 | 15 |
| Part 1 | 2 min 55 sec | 1 min 36 sec | 1 min 18sec |
| Part 2 | 9 min 47 sec | 9 min | 8 min 13 sec |
| Part 3 | 30 min | 24min | 20min |

The above statistics are for medium file. The execution for large dataset is taken for part 1 and part 2. But for part 3 , the time taken for execution is going to a huge value. The outputs for part1 and part 2 of large data file and output for all parts of medium file can be observed in the cluster.

The y-axis shows the time for execution and x-axis shows the different executors.

**Findings:**

The execution of a job depends on the availability of resources on the cluster. For example, the amount of time required to execute the application implemented for third question should be more than the first two parts based on the measurements involved in the process. As the number of executors increases, the time for execution decreases. The execution time also depends on the availability of cluster. The number of users in the cluster increases, the execution time goes on increasing and vice-versa. Thus, we can say that the number of jobs running on the cluster influences the execution.

The execution time depends on following factors:

* Number of jobs running on cluster
* Number of reducers used
* Amount of calculations or complexity of code involved