Big Data Analytics

Assignment-3

Chethana Dukkipati

1637386

Problem Description:

The problem in our assignment is to see the differences in terms of file sizes and time of execution when the output is stored in different file formats.

We are provided with a huge dataset with multiple documents in which we need to find the similarity between two documents. So as part of one needs to find the inverted index for the most occurred word in the documents where inverted index is the division of number of times a word occurred in a document to the total number of documents. Thus, inverted index needs to be find for each word in all the documents.

Once inverted index is obtained, one needs to calculate the similarity index between two documents and store the results in a file.

The above process is performed as part of previous assignment by using spark. In this assignment one needs to store the results obtained after calculating inverted index in different file formats like Text, Avro, Parquet, Snappy Compressed file and read the files as part of finding similarity index between the documents and store the result in the same file formats.

The storage space and execution time of each file format is observed for 5 executors for different runs on a medium dataset and analyze the results.

Solution Strategy:

The entire assignment is divided into 5 sections where the first part is to implement the inverted indices and similarity index for the documents and store them in a text file format. The text files are already implemented as part of second assignment and are directly executed for this part.

For other file formats like avro, the procedure I followed to derive them from the available data is same as that of the one in second assignment. But at the ending of the program for implementing the inverted indices, I converted the obtained resultant RDD into data frame using “toDF” function with column names specified. This data frame is written into the text files using write.format function and saving them using save option as below:

dataframe\_result.write.format("com.databricks.spark.avro").save("/bigd11/outputfilename.avro")

Thus the result of inverted indices part is stored in avro file and the same is being read for the similarity indices part. The read can be performed by using :

sqlContext.read.format("com.databricks.spark.avro").load("/bigd11/outputavrofile")

through which I obtained a dataframe and is converted to RDD by using dataframe.rdd function on which the usual process is carried out.

I have used ZIP functionality to derive the values for the documents and to get the similarity function. These obtained similarity indices are again converted into dataframe and is written to the text file as done earlier.

The same procedure can be followed to parquet files except the syntax for writing and reading the files.

For Parquet files, it is more simple than the avro files, the writing can be done by

dataframe\_result.write.parquet("/bigd11/outputfilename.parquet”)

and reading can be performed by

sqlContext.read.parquet(“outputfilepath.parquet”)

For the fourth part, I have done snappy compression format for the parquet file. The snappy conversion for a parquet file can be performed by adding sqlContext.setConf("spark.sql.parquet.compression.codec", "snappy") in the program and writing and reading can be performed same as parquet files.

Writing can be done through: dataframe\_result.write.parquet("/bigd11/outputfilename”)

and reading can be performed by: sqlContext.read.parquet(“outputfilepath”)

In the last part, I have collected size of the files and time taken for execution for all the different file formats in 3 attempts and formulated a table to compare them which is presented in the results part.

Mainly in the entire program implementation, I have used dataframe to rdd conversion and vice versa, writing and reading to different file formats. The entire program to calculate the inverted index and similarity index is based on pyspark. For calculating similarity indices pairwise in documents, I have taken ZIP command as the base.

The program for storing the results in avro formats is given in question22.py and question32.py

The program for storing results in parquet format is given in parquet2.py and paruqet3.py an fir snappy compression is given in parquetsnappy2.py and parquetsnappy3.py

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. For writing we have used different packages like spark-avro, spark-parquet. To read the rdd data into dataframe, sql context and sql are imported. For snappy compression we have used compression codec function is called through sqlcontext conf setting is used.

Description of measurements performed:

I have performed the observations on medium dataset for 5 executors. I have tried to execute the programs for three times to understand the performance of files and its dependability on characterisstics like size and time of execution.

The file size can be retrieved by executing command “hdfs dfs -du -h /bigd11/”. This gives the disk space consumed by the file without replication and with replication. The time for execution is calculated from the cluster page.

Results:

The values for different executions are shown in the tables.

The sizes of different file formats for different runs and 5 executors for inverted index are given in the below table. The size is expressed in MB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Runs | Text | Avro | Parquet | Snappy Compression(Parquet) |
| Run1 | 18.3 | 5.8 | 2.8 | 2.8 |
| Run2 | 18.3 | 5.8 | 2.8 | 2.8 |
| Run3 | 18.3 | 5.8 | 2.8 | 2.8 |

The sizes of different file formats for different runs and 5 executors for similarity indices are given in the below table. The size is expressed in MB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Runs | Text | Avro | Parquet | Snappy Compression(Parquet) |
| Run1 | 15.6 | 4.8 | 2.7 | 2.6 |
| Run2 | 15.6 | 4.8 | 2.7 | 2.7 |
| Run3 | 15.6 | 4.8 | 2.7 | 2.7 |

The run times for the different file formats for different runs and 5 executors for inverted index are given as in minutes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Runs | Text | Avro | Parquet | Snappy Compression(Parquet) |
| Run1 | 9 min 30sec | 9 min | 7min 2sec | 7 min 9sec |
| Run2 | 9 min | 8min 54sec | 7min 4 sec | 7min 9sec |
| Run3 | 9 min 30 sec | 8min 44sec | 7min 7 sec | 7 min 8sec |

The run times for the different file formats for different runs and 5 executors for similarity index are given as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Runs | Text | Avro | Parquet | Snappy Compression(Parquet) |
| Run1 | 30 min | 4 min 1sec | 3 min 15sec | 3min 10sec |
| Run2 | 30min 4 sec | 4 min 1sec | 3min 25sec | 3min 14sec |
| Run3 | 30 min | 4 min | 3 min 25 sec | 3 min 45 sec |

**Graphs:**

The x-axis shows the number of runs for different file formats and Y-axis their corresponding File Sizes

The x-axis shows the number of runs for different file formats and Y-axis their corresponding execution Times.

**Findings:**

The execution of a job depends on the availability of resources on the cluster. 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 file size of parquet file is less than all the other file formats for both the parts Inverted Index and Similarity Index. The time of execution is also less for parquet files.

For different runs the size of file is same whereas the time of execution differs from different runs. The snappy compression file size and parquet files are almost same, and the execution time is also very less.

The time taken for executing text file for similarity index is more but whereas for next file formats it is less than the inverted index.

The mean values for size of files are as follows in order of text, avro, parquet, snappy

18.3, 5.8, 2.8 and 2.8 for inverted index

15.6, 4.8, 2.7 and 2.7 for similarity index

The mean values for time of execution of files are as follows in order of text, avro, parquet, snappy

7.33, 8.87, 7.06, 7.14 for inverted index

30.02, 4.006, 3.35, 3.38 for similarity index

The mean values for time of execution for files as follows:

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
* The file size also effects the execution time.