Problem Description:

The solution of the problem is divided into two parts and two lists of flight data are provided for carrying out the task.

Part 1: Write a pyspark code that converts a csv files into parquet, sequence, and json file and compare the size with the input file.

Part 2: Write a separate *pyspark* code for each of the input file format produced in the first part along with csv and calculate the percentage of delayed flights per Origin Airport. Next compare the execution time for each input file format by varying the number of executer.

Output of these experiments are further analyzed using graphs.

Solution Strategy:

Part 1: Part 1 to convert csv files into different format started by setting an app name to *SparkContext* which is accessible via sc variable in Python. To access the relational functionality in Spark the *SQLContext* class is used here. Next list of files was read using *textFile* and RDD of tuples were created. Then the RDD was further converted into a Dataframe of appropriate columns and finally the file was saved in Parquet format in HDFS. To create a JSON file we saved the same data frame into a JSON file. To create the sequence file, we read the input files into a RDD and mapped each line with an empty key and value pair. And then saved the RDD into a Sequence File.

Part 2: In part 2, I started by setting executor memory to 4gb and giving distinct app name for each of the 4 file formats and set the SQL context on SparkContext object. airlineTuple function is created which splits a line into a tuple of multiple values and it is used only while reading sequence and parquet files. The input file is read from the defined location into RDD using sparkcontext with textfile method. Then a DF was created with appropriate column names and further filtered into a new data frame by excluding NA values to facilitate further calculation on it. Since a DataFrame is a dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in Python. Hence, further SQL like queries were performed on the filtered data frame to calculate the final percentage of the delayed flight per origin.

Finally, all the programs for different input file format were run for varying number of executors as in 5, 10 and 15. Results were documented for each run and graphs were plotted.

Resources Used:

Putty: PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. PuTTY is open source software that is available with source code and is developed and supported by a group of volunteers.

PySpark: Apache Spark provides APIs in non-JVM languages such as Python. Pyspark is a python API and it paves way to the spark programming model to python. It supports almost all the counterpart function in the Scala environment and uses the Python functions and returns the python collection types.

Whale Cluster:
The whale cluster specifications are as follows:

Technical Data
- 57 Approx. 1522H nodes (whale-001 to whale-057)
- Two 2.2 GHz quad-core AMD Opteron processor (8 cores total)
- 16 GB main memory
- Gigabit Ethernet
- 4xDDR InfiniBand HCAs (unused at the moment)

Network Interconnect
- 144 port 4xInfiniBand DDR Voltaire Grid Director ISR 2012 switch (donation from TOTAL, shared with crill)
- Two 48 port HP GE switch

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

MS Office and Python were used along with the other resources.

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 File*
-rw-r--r-- 1 bigd27 hadoop 1800 May  3 15:43 File_Converter.py
bigd27@whale:~> ll part2_*
-rw-r--r-- 1 bigd27 hadoop 2170 May  5 07:42 part2_csv.py
-rw-r--r-- 1 bigd27 hadoop 1051 May  4 08:47 part2_json.py
-rw-r--r-- 1 bigd27 hadoop 1059 May  5 07:41 part2_parq.py
-rw-r--r-- 1 bigd27 hadoop 2179 May  5 06:07 part2_seq.py
```

- Below is the command used to run every job.

spark-submit --master yarn part2_seq.py /bigd27/SequenceFile/part /bigd27/output*

Vary the number of executor 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.

Results:

Part1 Results:
Below is the command used to determine the file size for each format.

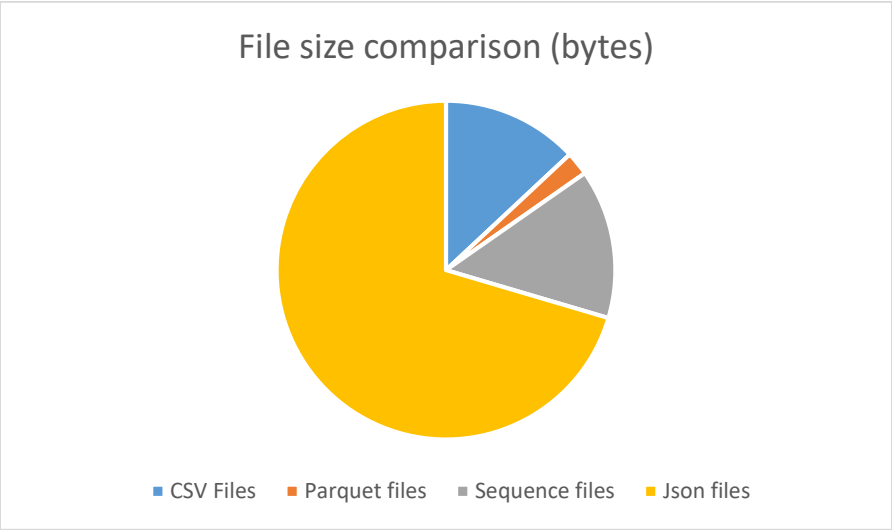
```
bigd27@whale:~> hdfs dfs -du -s /bigd27/SequenceFile
3728289531  /bigd27/SequenceFile

bigd27@whale:~> hdfs dfs -du -s /bigd27/ParquetFile
591382025  /bigd27/ParquetFile

bigd27@whale:~> hdfs dfs -du -s /bigd27/JsonFile
18400905660  /bigd27/JsonFile

bigd27@whale:~> hdfs dfs -du -s /cosc6339_s17/flightdata-full
3405266011  /cosc6339_s17/flightdata-full
```

Comparison	CSV Files	Parquet files	Sequence files	Json files
File size (bytes)	3405266011	591382025	3728289531	18400905660



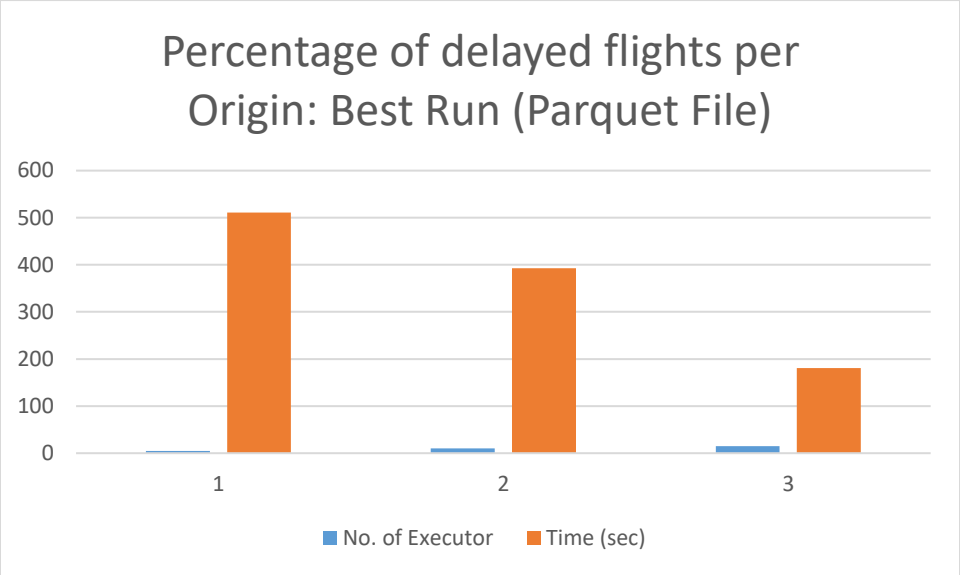
After conversion, each of file format was compared and Parquet file occupied the least disk space and Json occupied the most disk space.

Part2 Results:

Parquet Files:

Below is the plot of best run which nothing but the minimum execution time for varying no. of executors.

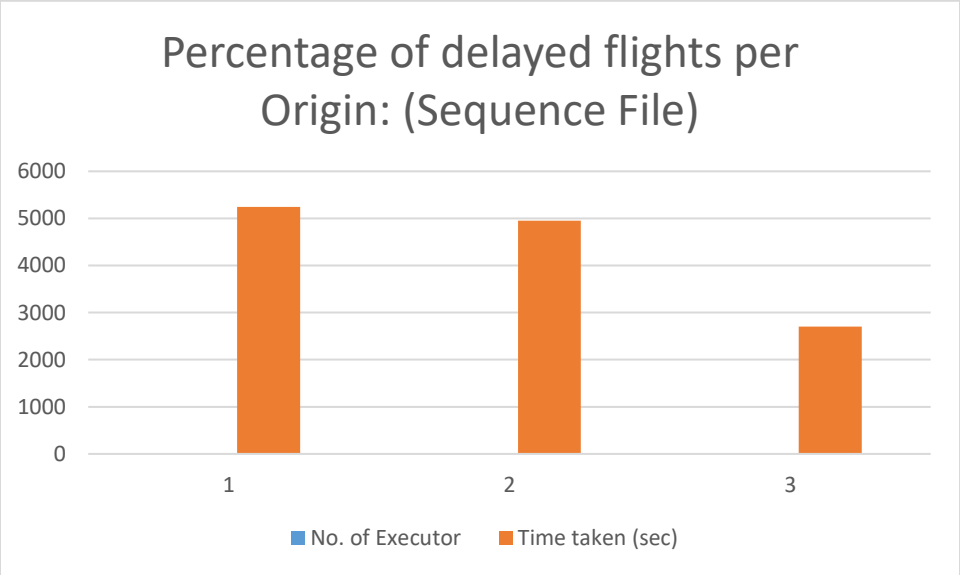
Parquet File	1st Run	2nd Run
No. of Executor	Time taken (sec)	Time taken (sec)
5	541	511
10	393	412
15	206	181



For sequence, csv and Json file format the execution time is substantially more. Hence execution could not be carried out multiple times.

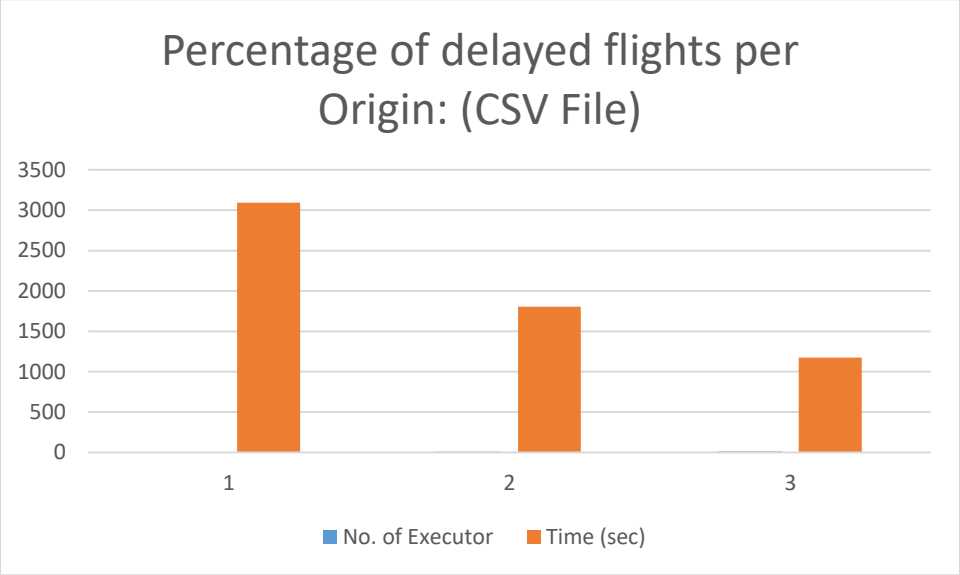
Sequence Files:

Sequence Files	1st Run
No. of Executor	Time taken (sec)
5	5243
10	4952
15	2701



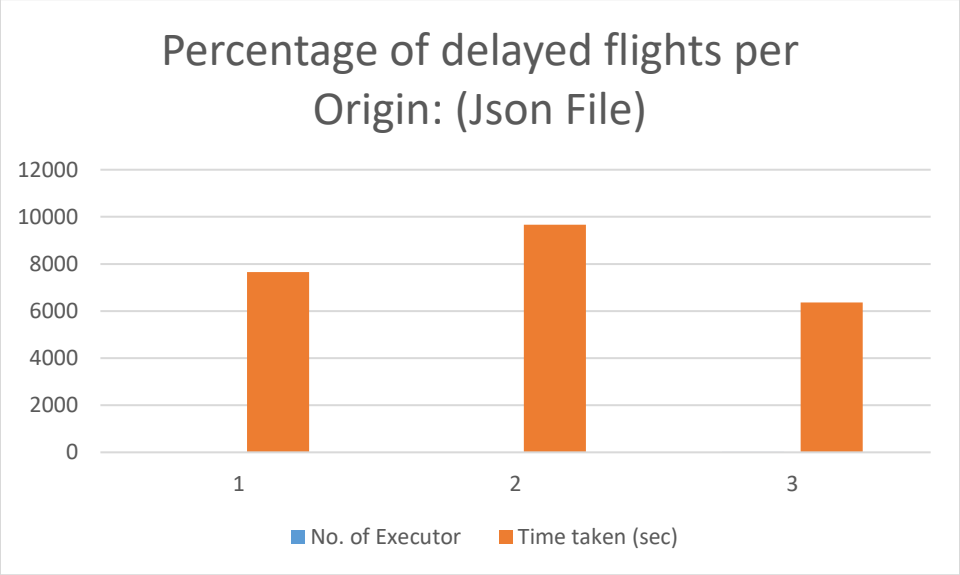
CSV Files:

CSV Files	1st Run
No. of Executor	Time taken (sec)
5	3092
10	1806
15	1173.6

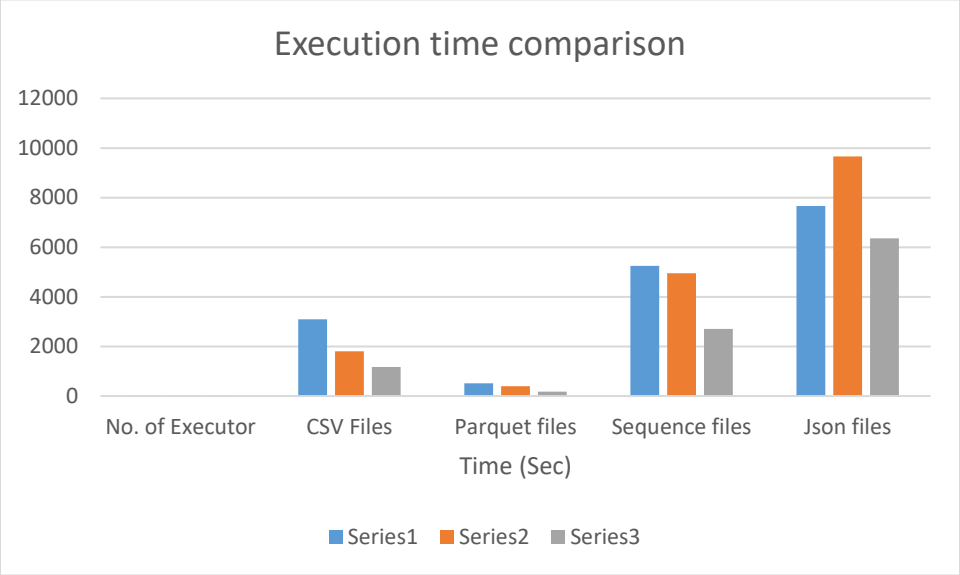


Json Files:

Json Files	1st Run
No. of Executor	**Time taken (sec)**
5	7654.5
10	9662.3
15	6364



Part 2	CSV Files	Parquet files	Sequence files	Json files
No. of Executor	Execution (Time sec)	Execution Time (sec)	Execution Time (sec)	Execution Time (sec)
5	3092	511	5243	7654.5
10	1806	393	4952	9662.3
15	1173.6	181	2701	6364



Observation:

From the above graph its evident that with the parquet file format we get the best execution time and it provides the best file compression. However, execution time for JSON is substantially more compared to other formats and gives the worst performance. Wherein CSV and Sequential file demonstrates moderate performance but can be considered poor as compared to Parquet file format. The percentage of delayed flights per Origin result can be found in HDFS at /bigd27/output.

References:

[1] http://diybigdata.net/2016/09/airline-flight-data-analysis-part-2-analyzing-on-time-performance/

[2] http://diybigdata.net/2016/08/airline-flight-data-analysis-data-preparation/

[3] http://www.thecloudavenue.com/2016/10/converting-csv-to-parquet-using-spark.html