Name: Pushpendra
PSID: 1639812
Server Account: bigd41
Assignment: COSC6397 HW3

Details of Input file and server used to complete this assignment:

Medium Input dataset for final submission: /cosc6339_hw2/gutenberg-500/

Server Info:

Server used: http://whale.cs.uh.edu
Technical detail of the whale server:

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)

Details link: http://pstl.cs.uh.edu/resources/whale

Pydoop version: 1.2.0
Python version: 2.7.13
Spark version: 2.2.2
Parquet version:1.8.2
Avro version:1.8.2

Problem Statement: Study the execution time and file size when creating the inverted index and similarity matrix using out of box spark API, Parquet , snappy Parquet and Avro.

General Description:

The report presents finding on the execution time and file size created for Inverted Index and Similarity matrix.

- Number of executors used to execute all the programs is 5.
- All the time are reported in seconds and file size in MB. File size is the sum of all file sizes created for specific task.
- Additionally, submitted an excel file for measurements which includes time and application id for executing the job.
- Inverted Index is the output of Question 2 and similarity matrix is result of Question 3.
- Spark data format is referred to as Out of box spark API in this report.
- The execution is repeated 3 times and referred to as Set 1 , Set 2 and Set 3.
- Commands to execute the program is mentioned in each question. The <attempt1> must be changed the attempt the user is making.
- Source code for HW3 is inside folder "SourceCode" and HW2 is in folder "Sourcecode/HW2". When executing the commands folder location must be changed.
- The output folders are created in the below location in HDFS:

  /bigd41/HW3/HW2/guten500/attempt* -- for OOTB spark format of HW2

  /bigd41/HW3/guten500/attempt* -- for parquet and avro format of HW3

a) Take your code from the second assignment, and derive the solution for part2 and part3 into separate files/programs.

–Part 2 writes the inverted index into a textfile

–Part 3 reads the inverted index from the textfile

Already submitted in two files as part of HW2. Following is the execution part:

- Commands for executing the program:

```
spark-submit --master yarn HW2_Q2.py 5 <attempt1>
spark-submit --master yarn HW2_Q3.py 5 <attempt1>
```

- Input files: HW2_Q2.py, HW2_Q3.py
- Output folders: Q2_invertedtable, Q3_similaritymatrix

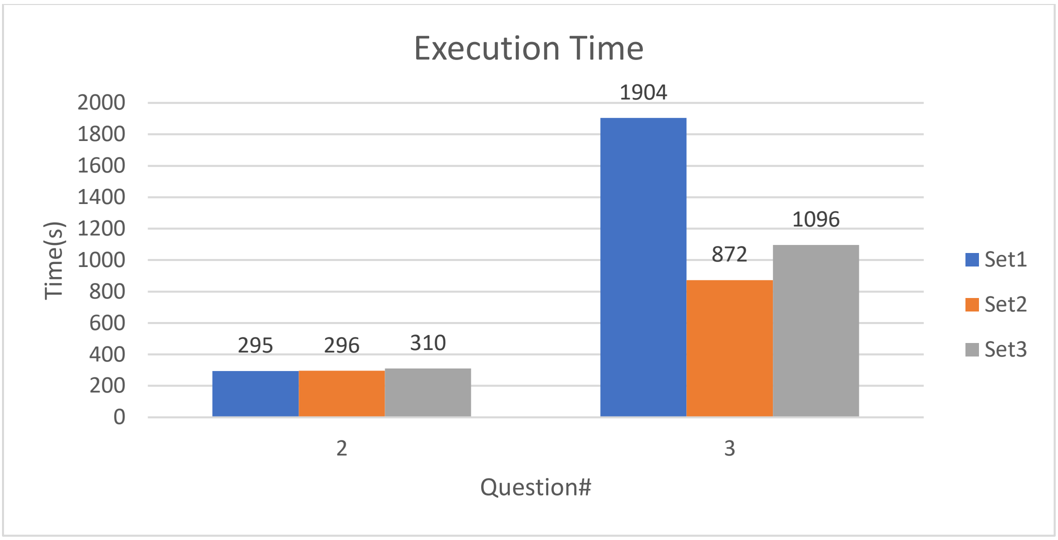
Result:

Measurement taken for Question 2 and Question 3 with Application Id and Start time of the program execution:

Quest#	Set 1	Set 2	Set 3
2	application_1542381842736_1146 Tue Nov 27 23:40:09 -0600 2018	application_1542381842736_1175 Wed Nov 28 16:03:40 -0600 2018	application_1542381842736_1186 Thu Nov 29 13:32:49 -0600 2018
3	application_1542381842736_1152 Tue Nov 27 23:47:35 -0600 2018	application_1542381842736_1176 Wed Nov 28 16:09:49 -0600 2018	application_1542381842736_1188 Thu Nov 29 13:38:25 -0600 2018

Time taken in seconds for the program to execute:

Question#	Set 1	Set 2	Set 3
2	295	296	310
3	1904	872	1096



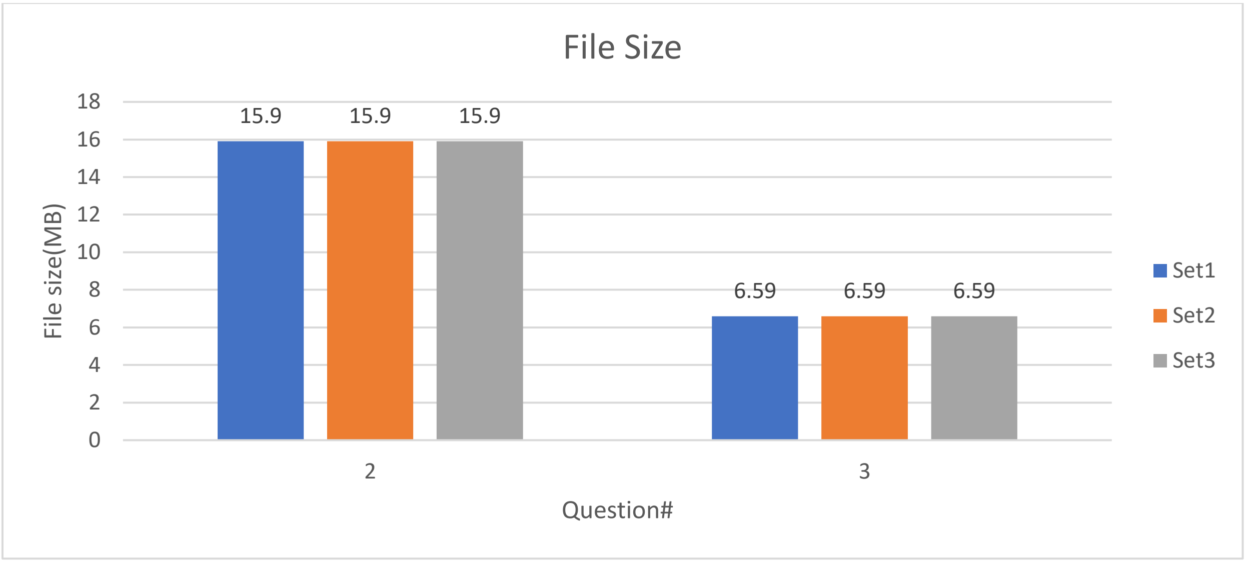
Analysis:

a) The execution time for inverted index remains almost same with very minute difference in all trail.

b) The execution time for similarity matrix differs and one of the reason may be other jobs running on the server where it is taking more time.

Size of the Inverted index and similarity matrix in MB:

Question#	Set 1	Set 2	Set 3
2	15.9	15.9	15.9
3	6.59	6.59	6.59



Analysis:
a) The file size created for both Inverted index as well as similarity matrix remains constant in all trail.

b) Implement a solution for Part 2 which writes the data using Avro, and Part 3 reads the inverted index as an Avro file and writes the final result as an Avro file.

- Solution strategy: To implement saving the data using Avro, we need to call format("com.databricks.spark.avro")").option("com.databricks.spark.avro","uncompressed") during saving the data. Similarly when reading the data we need to spark.read.format("com.databricks.spark.avro") before loading the data.
Here avro is being used without compression.

  For example:

  ```
  invertedTable.toDF().write.format("com.databricks.spark.avro").option("com.da
  tabricks.spark.avro","uncompressed").save(invertedAvroFilePath)
  ```

  ```
  spark.read.format("com.databricks.spark.avro").load(invertedTableInputPathAv
  ro)
  ```

- Commands for executing the program:

  ```
  spark-submit --master yarn HW3_Q2_Avro.py 5 <attempt1>
  spark-submit --master yarn HW3_Q3_Avro.py 5 <attempt1>
  ```

- Script file name: HW3_Q2_Avro.py, HW3_Q3_Avro.py
- Output folders: Q2_avro, Q3_similaritymatrixAvro

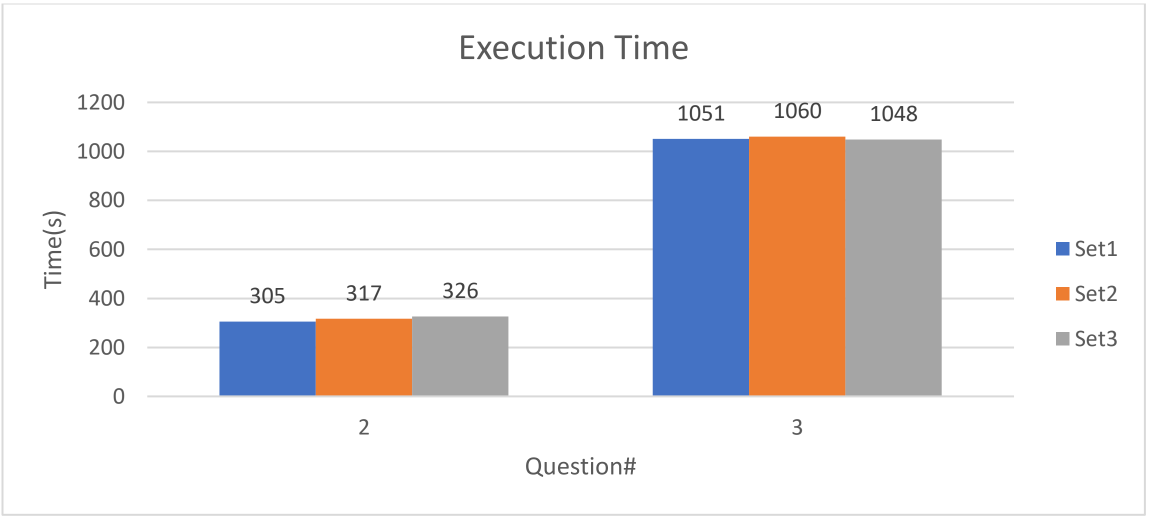
Result:

Measurement taken for Question 2 and Question 3 with Application Id and Start time of the program execution:

Question#	Set 1	Set 2	Set 3
2	application_1542381842736_1159 Wed Nov 28 00:22:38 -0600 2018	application_1542381842736_1178 Wed Nov 28 16:43:04 -0600 2018	application_1542381842736_1187 Thu Nov 29 13:35:18 -0600 2018
3	application_1542381842736_1160 Wed Nov 28 00:29:08 -0600 2018	application_1542381842736_1179 Wed Nov 28 16:48:57 -0600 2018	application_1542381842736_1189 Thu Nov 29 13:42:52 -0600 2018

Time taken in seconds to execute each of the program:

Question#	Set 1	Set 2	Set 3
2	305	317	326
3	1051	1060	1048



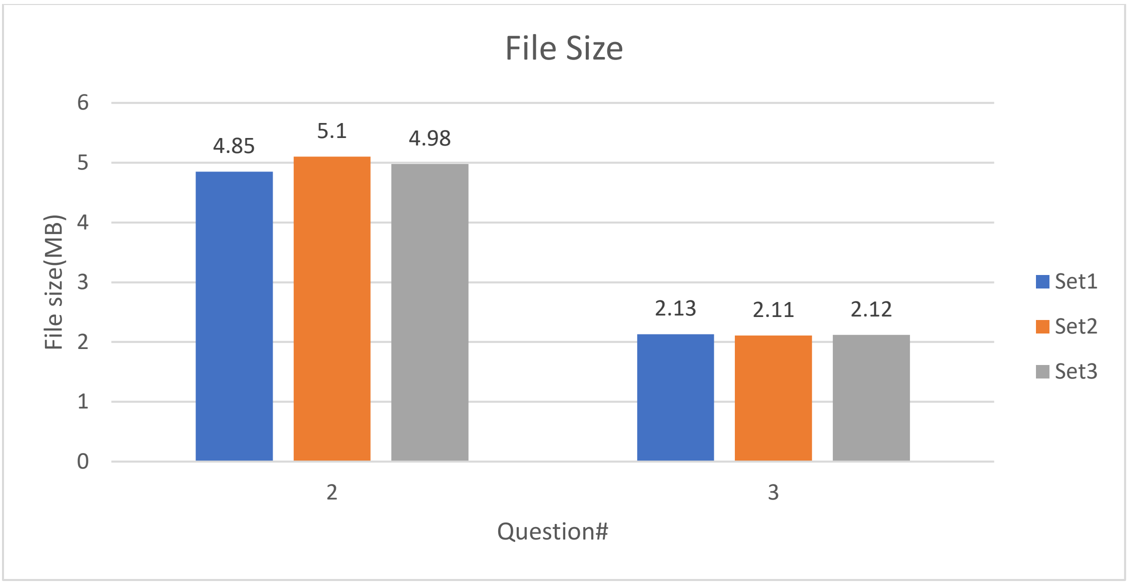
Analysis:

a) The execution time of inverted index using avro varies from 305 to 326. One of the reason for this may be the overload or other jobs executed by the server.

b) The execution time for creating similarity matrix is much closer to each other.

Size of the Inverted index and similarity matrix in MB:

Question#	Set 1	Set 2	Set 3
2	4.85	5.10	4.98
3	2.13	2.11	2.12



Analysis:

a) The file size created for both Inverted index as well as similarity matrix varies by a very small margin.

c) Same as part b, but using parquet files.

- Solution strategy: To implement saving the data using Parquet, we need to call write.option("compression", "none") during saving the data. Similarly when reading the data we need to spark.read.parquet() before loading the data. Here parquet is used without compression. In implementation, the program assumes that there will be 3 args values to be passed from shell. So there is only one file for Parquet with compression and without compression and using if else condition both cases are handled.

  For example:

```
invertedTable.toDF().write.option("compression",
"none").parquet(invertedUncompressedParquetFilePath)
```
```
spark.read.parquet(invertedTableInputPathParq)
```

- Commands for executing the program:

```
spark-submit --master yarn HW3_Q2_Parquet.py 5 <attempt1>
spark-submit --master yarn HW3_Q3_Parquet.py 5 <attempt1>
```

- Script file name: HW3_Q2_Parquet.py, HW3_Q3_Parquet.py
- Output folders: Q2_parque_uncompressed, Q3_similaritymatrixParq_uncompressed

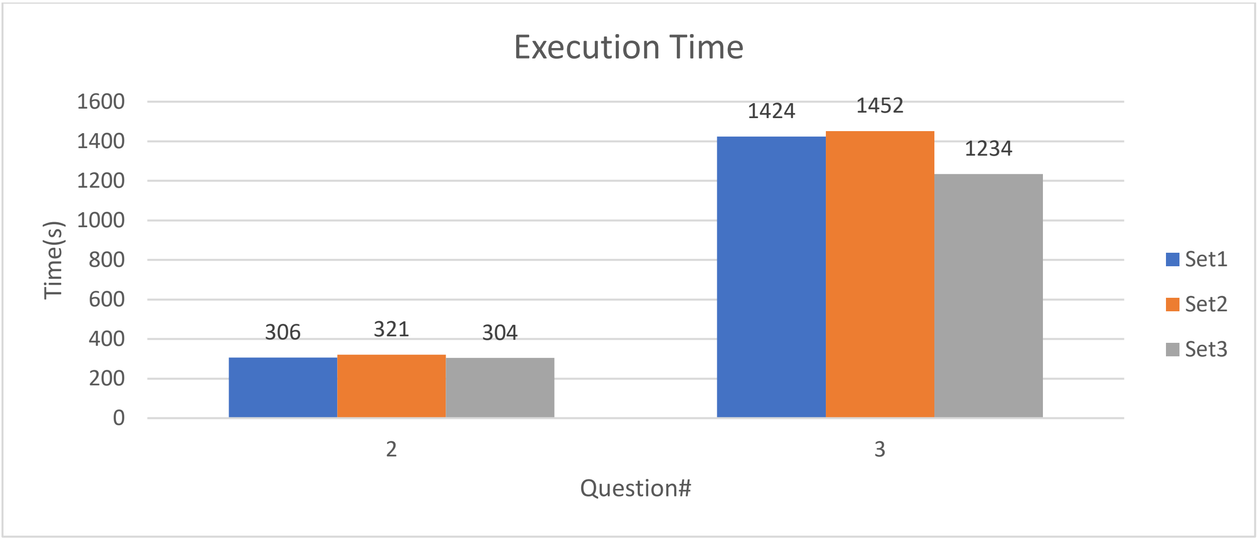
Result:

Measurement taken for Question 2 and Question 3 with Application Id and Start time of the program execution:

Question#	Set 1	Set 2	Set 3
2	application_1542381842736_1166 Wed Nov 28 00:47:55 -0600 2018	application_1542381842736_1182 Wed Nov 28 17:08:13 -0600 2018	application_1542381842736_1190 Thu Nov 29 13:59:07 -0600 2018
3	application_1542381842736_1214 Fri Nov 30 01:04:43 -0600 2018	application_1542381842736_1216 Fri Nov 30 01:33:04 -0600 2018	application_1542381842736_1218 Fri Nov 30 02:00:41 -0600 2018

Time taken in seconds to execute each of the program:

Question#	Set 1	Set 2	Set 3
2	306	321	304
3	1424	1452	1234



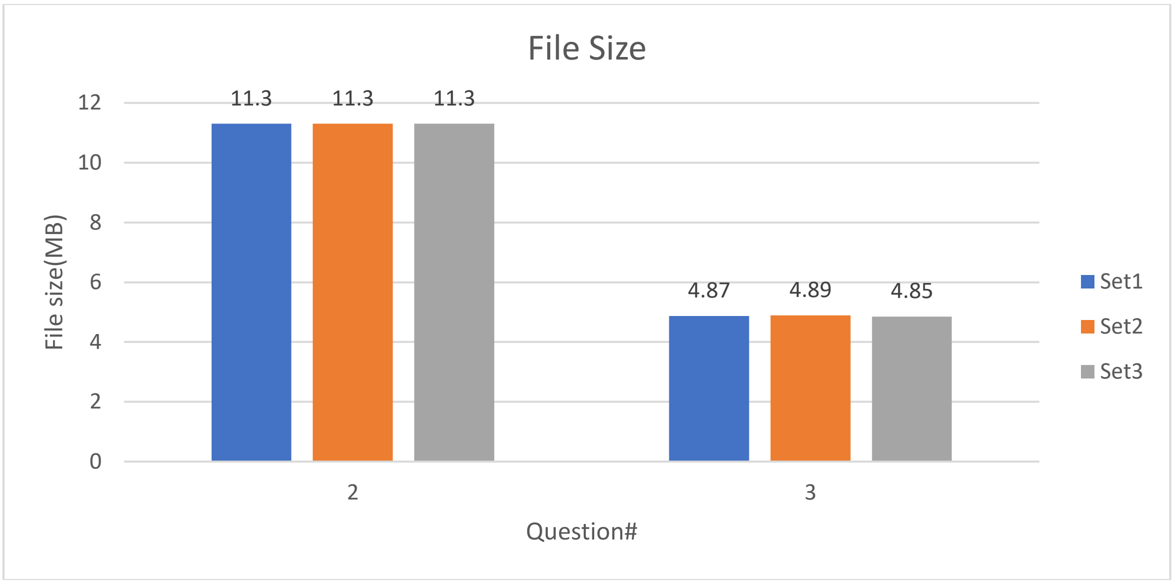
Analysis:

a) The execution time of inverted index varies from 306 to 321. One of the reason for this may be the overload or other jobs executed by the server.

b) The execution time for creating similarity matrix varies a bit more. First two attempts they are very near to each other while in third case it takes much less time. It may be due to less load on the server.

Size of the Inverted index and similarity matrix in MB:

Question#	Set 1	Set 2	Set 3
2	11.3	11.3	11.3
3	4.87	4.89	4.85



Analysis:

a) The file size created for both Inverted index remains constant whereas for similarity matrix varies by a very small margin.

b) Parquet may change the order of rows so if the sorted data is being written, then it is must to sort again after reading the Parquet file to guarantee that your program dependent on this output assumes data to be sorted. Other way can be that write unsorted data and after reading sort it.

d) Implement a version of either Part b or Part c (its your chose) that creates snappy-compressed files (either Avro or Parquet).

   Implementing the solution using Parquet snappy-compressed files.

- Solution strategy: To implement saving the data using Parquet, we need to call write.option("compression", "snappy") during saving the data. Similarly when reading the data we need to spark.read.parquet() before loading the data. Here snappy compression is used. In implementation, the program assumes that there will be 4 args values to be passed from shell. So there is only source code file for Parquet with compression and without compression and using if else condition both cases are handled.

  For example:

invertedTable.toDF().write.option("compression", "snappy").parquet(invertedCompressedParquetFilePath)
spark.read.parquet(invertedTableInputPathParq)

- Commands for executing the program:

```
spark-submit --master yarn HW3_Q2_Parquet.py 5 <attempt1> 1
spark-submit --master yarn HW3_Q3_Parquet.py 5 <attempt1> 1
```

- Script file name: HW3_Q2_Parquet.py, HW3_Q3_Parquet.py
- Output folders: Q2_parque_compressed ,Q3_similaritymatrixParq_compressed

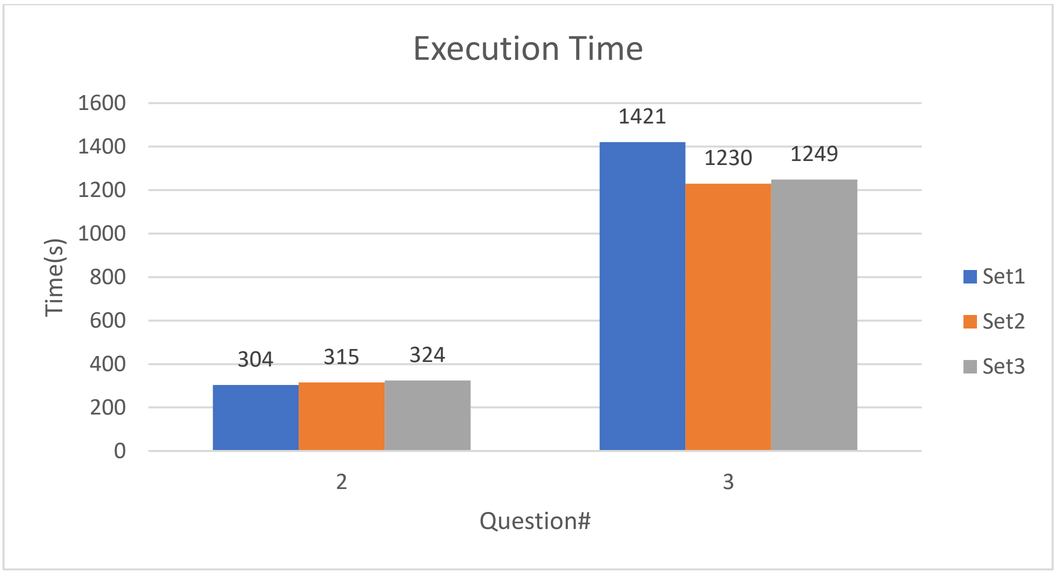
Result:

Measurement taken for Question 2 and Question 3 with Application Id and Start time of the program execution:

Question#	Set 1	Set 2	Set 3
2	application_1542381842736_1168 Wed Nov 28 01:13:17 -0600 2018	application_1542381842736_1180 Wed Nov 28 16:58:29 -0600 2018	application_1542381842736_1191 Thu Nov 29 14:00:57 -0600 2018
3	application_1542381842736_1215 Fri Nov 30 01:06:36 -0600 2018	application_1542381842736_1217 Fri Nov 30 01:35:24 -0600 2018	application_1542381842736_1219 Fri Nov 30 02:02:10 -0600 2018

Time taken in seconds to execute each of the program:

Question#	Set 1	Set 2	Set 3
2	304	315	324
3	1421	1230	1249



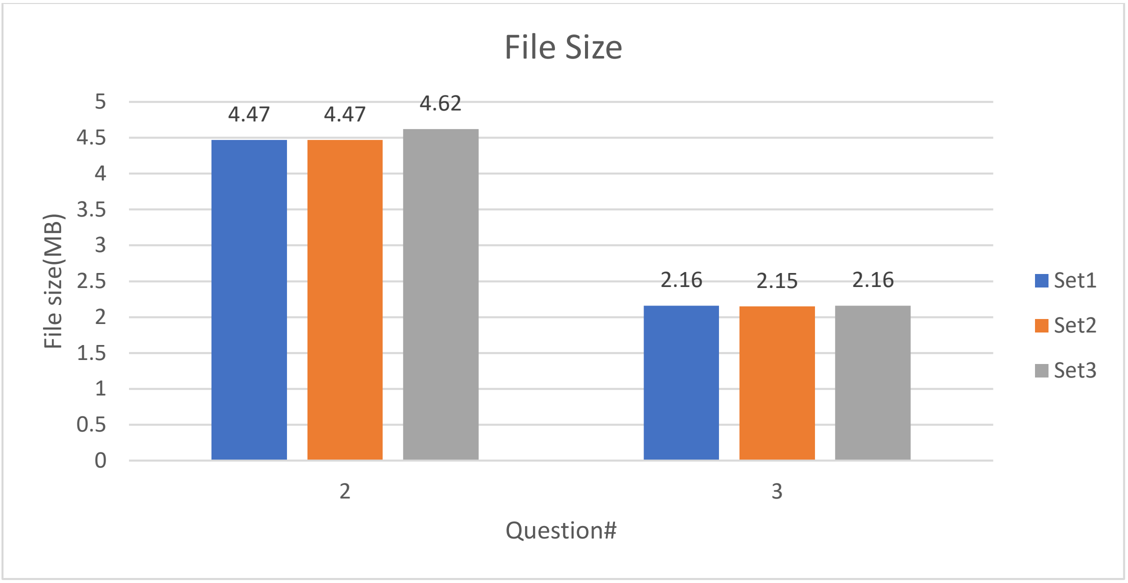
Analysis:

a) The execution time of inverted index varies from 304 to 324. One of the reason for this may be the overload or other jobs executed by the server.

b) The execution time for creating similarity matrix varies from 1230 to 1421. One of the reason for this may be the overload or other jobs executed by the server.

Size of the Inverted index and similarity matrix in MB:

Question#	Set 1	Set 2	Set 3
2	4.47	4.47	4.62
3	2.16	2.15	2.16



Analysis:

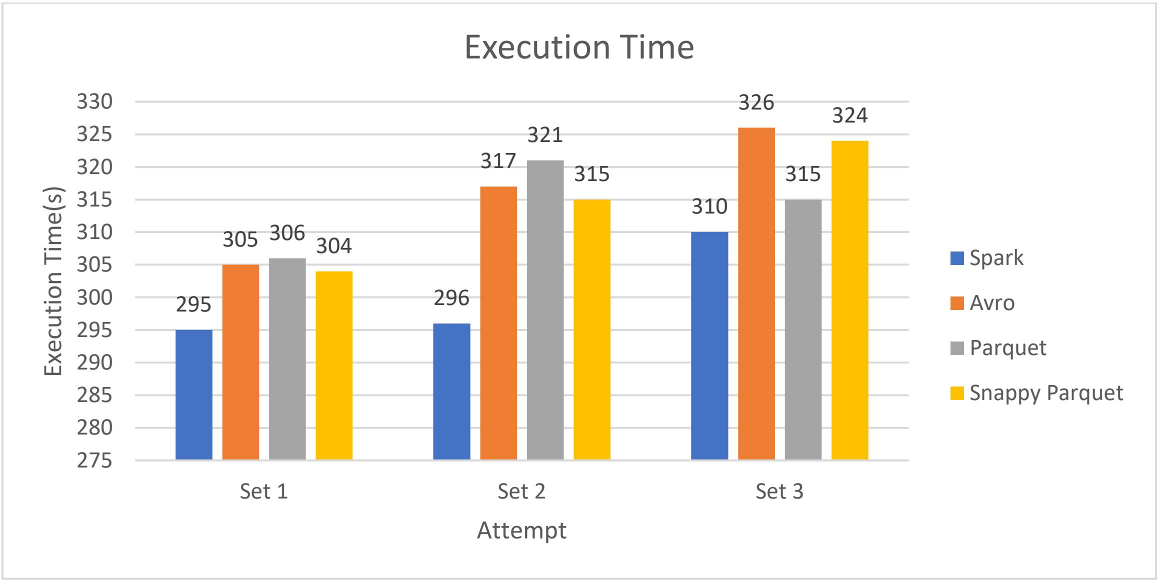
a) The file size created for both Inverted index as well as similarity matrix varies by a very small margin.

b) Parquet may change the order of rows so if the sorted data is being written, then it is must to sort again after reading the Parquet file to guarantee that your program dependent on this output assumes data to be sorted. Other way can be that write unsorted data and after reading sort it.

e) Compare file sizes for Parts a –d, as well as execution time of all versions of Part a-d using the corresponding format using the MEDIUM data set using 5 executors

Execution time in seconds for Inverted Index using all data format:

	Spark	Avro	Parquet	Snappy Parquet
Set 1	295	305	306	304
Set 2	296	317	321	315
Set 3	310	326	304	324



Analysis:

a) The time for execution is minimum in case of OOTB spark API program.

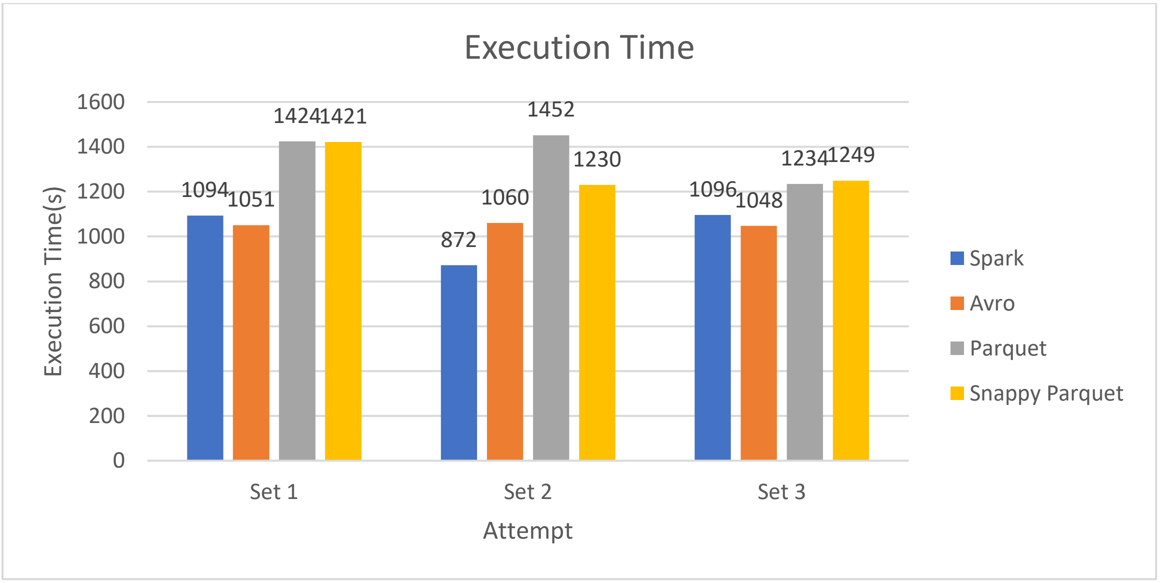
b) Comparing the time taken for avro and parquet with without compression, it is difficult to say which one is faster because sometimes avro has taken less and other case parquet.

c) Comparing the time for parquet with and without compression, the result does not clearly indicate which is slower or faster.

d) Comparing the time taken for avro and parquet with compression, its always parquet has taken less time.

Execution time in seconds for Similarity matrix using all data format:

	Spark	Avro	Parquet	Snappy Parquet
Set 1	1094	1051	1424	1421
Set 2	872	1060	1452	1230
Set 3	1096	1048	1234	1249



Analysis:

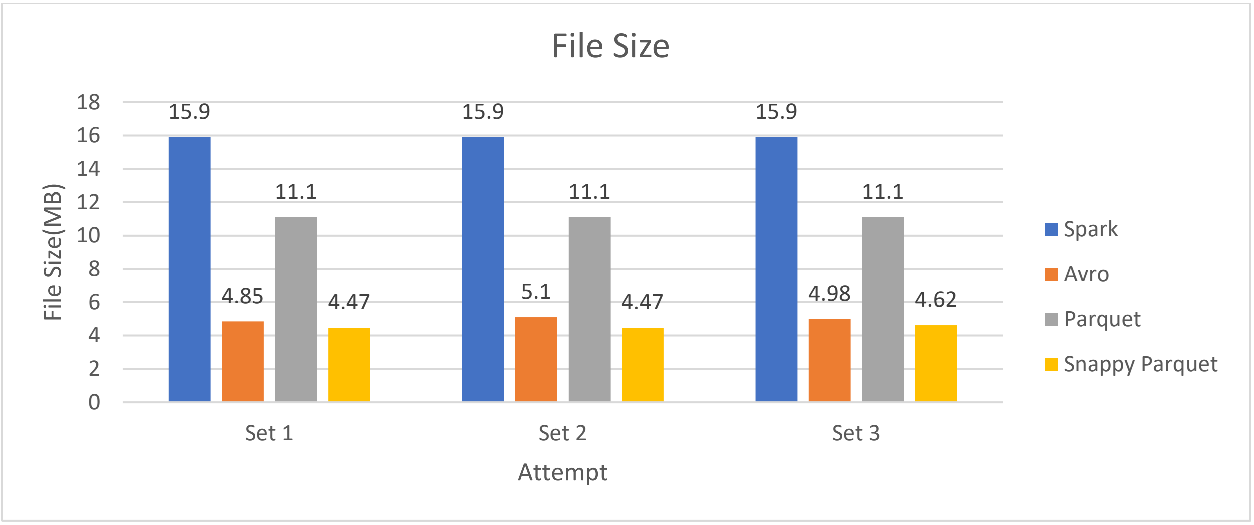
a) In general the time for execution is minimum in case of OOTB spark API. This must also be theoretically correct but as observed in above measurement there are few exceptions where it is not true. It may be because at that instant the sever may be processing other jobs as well impacting the time of execution for Spark data format.

b) Comparing the time taken for avro and parquet, in this case Avro has always taken less time.

c) Comparing the time taken for parquet with and without compression, the measurement does not give a clear picture which is faster or slower.

File size for each of the data format for Inverted Index:

	Spark	Avro	Parquet	Snappy Parquet
Set 1	15.9	4.85	11.3	4.47
Set 2	15.9	5.10	11.3	4.47
Set 3	15.9	4.98	11.3	4.62



Analysis:

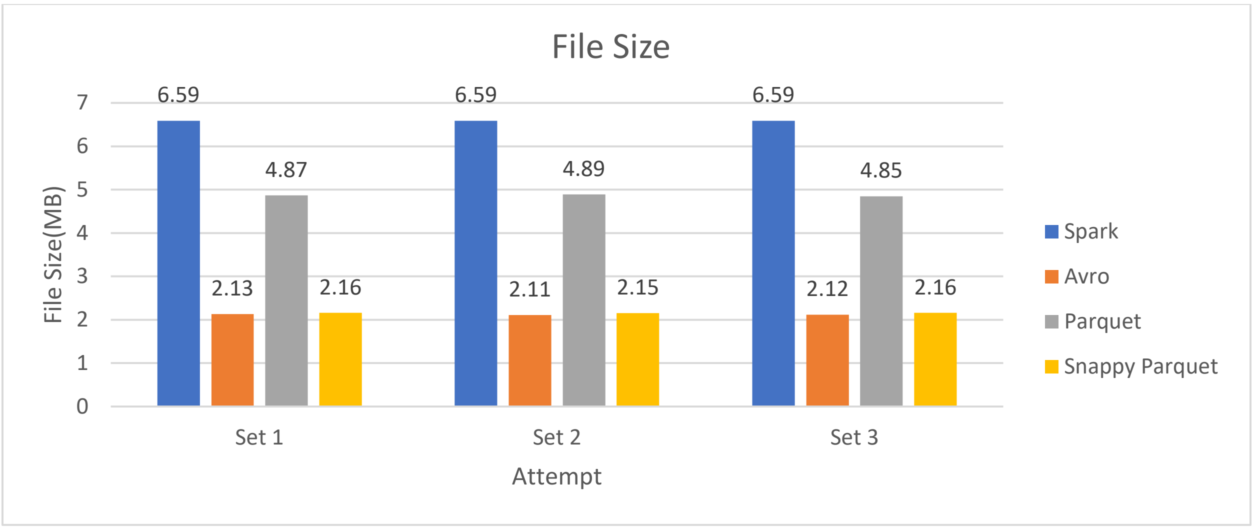
a) File size in case of OOTB spark is constant and is of largest size.

b) Minimum file size is in case of Snappy parquet.

c) It is better to use Avro or snappy parquet for compression as both of them have significant compression.

File size for each of the data format for Similarity matrix:

	Spark	Avro	Parquet	Snappy Parquet
Set 1	6.59	2.13	4.87	2.16
Set 2	6.59	2.11	4.89	2.15
Set 3	6.59	2.12	4.85	2.16



Analysis:

a) File size in case of OOTB spark is constant and is of largest size.

b) Minimum file size is in case of Avro.

c) It is better to use Avro or snappy parquet for compression as both have significant compression.

Conclusion:

1) OOTB Spark data format is fast in most of the cases with the tradeoff that it does occupy maximum file size.

2) With parquet and Avro, they take more time to process but then they take much less space than the Spark data format.