Programming Assignment-III
COEN 242, BiG Data

Anjaly George
Syeda Gousia Sultana

Due date: 06/13/2020

Contents In the document

PART 1

Problem Statement-1:

Rank the movie by their number of reviews(popularity) and select the top 10 highest ranked movies.

Implementation:

We have designed and implemented an efficient pyspark code using dataframes to select top 10 highest ranked movies given in a dataset. We have targeted towards obtaining the result with the least possible execution time and have come up with the best approach for efficient performance on our computers.

By trying to implement various approaches we have come to the conclusion that using various tuning parameters have improved performance. During the testing phase we have seen an improvement in performance by adjusting the number of partitions and implementing cache improved it further. We have implemented the code with multiple functions.

The ‘getOrCreate’ method as the name describes gets an existing spark session if there is an ongoing one else it will create a new one and assign it as the global default. We have created the dataframe by implementing the ‘StructType’ class and ‘StructField’ objects on the movies.csv and reviews.csv file respectively for creating a schema for the dataframe. The repartitions have been set to 1 after experimenting various values. We have noted that by increasing the value there was delay in the execution time which is probably because it is a small file which can be handled in few nodes and that the time taken to merge these shuffled files was more.

After reading and partitioning the file between 2 cores we have performed the groupby function to group all the identical values, count to count the number of times each value has appeared, orderby to sort them in an order and applied a limit of 10 on the descending order to get the top 10 reviews. We are storing all this in a cache to improve the throughput. We perform the join function as the last operation (to minimize the rows that are concatenated) on both the datasets to sort them by their corresponding columns using the orderBy function.

Problem Statement-2:

Find all the movies that its average rating is greater than 4 stars and also each of these movies should have more than 10 reviews

Implementation

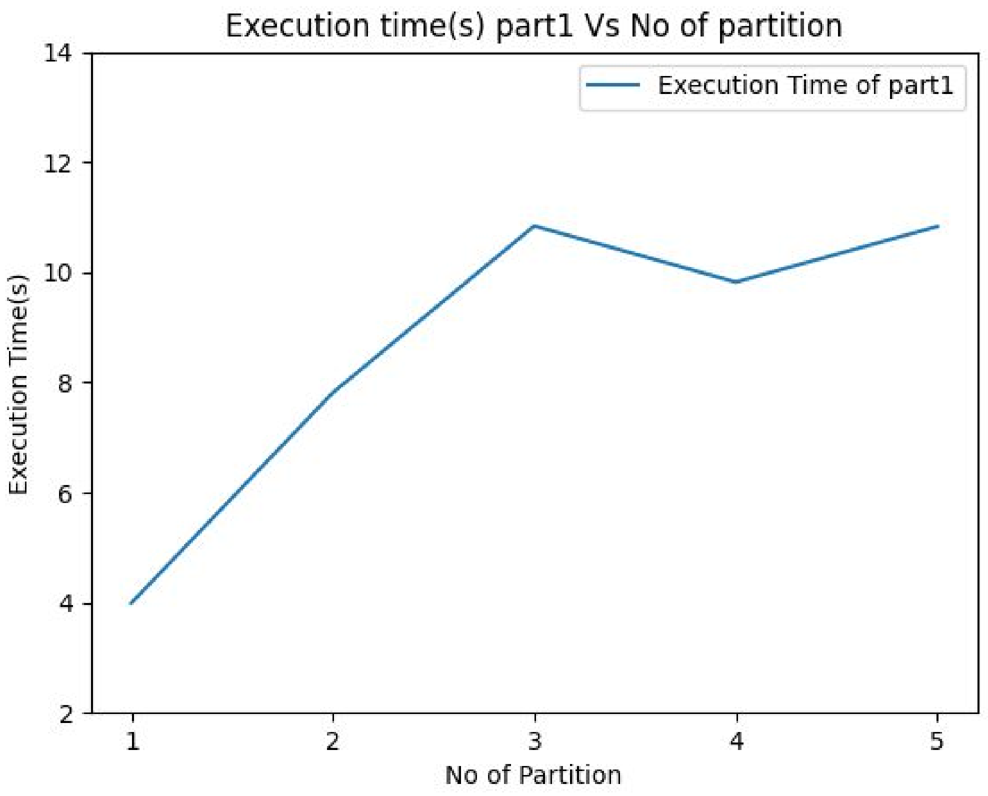
For the second part we performed aggregate function on the ratings to compute the average and return it as a dataframe . Using filter function filtered the movies with ratings greater than 4 and whose reviews are more than 10. Here implemented the cache method so that the performance will be increased as the join, select and orderBy need to be performed for returning the result.

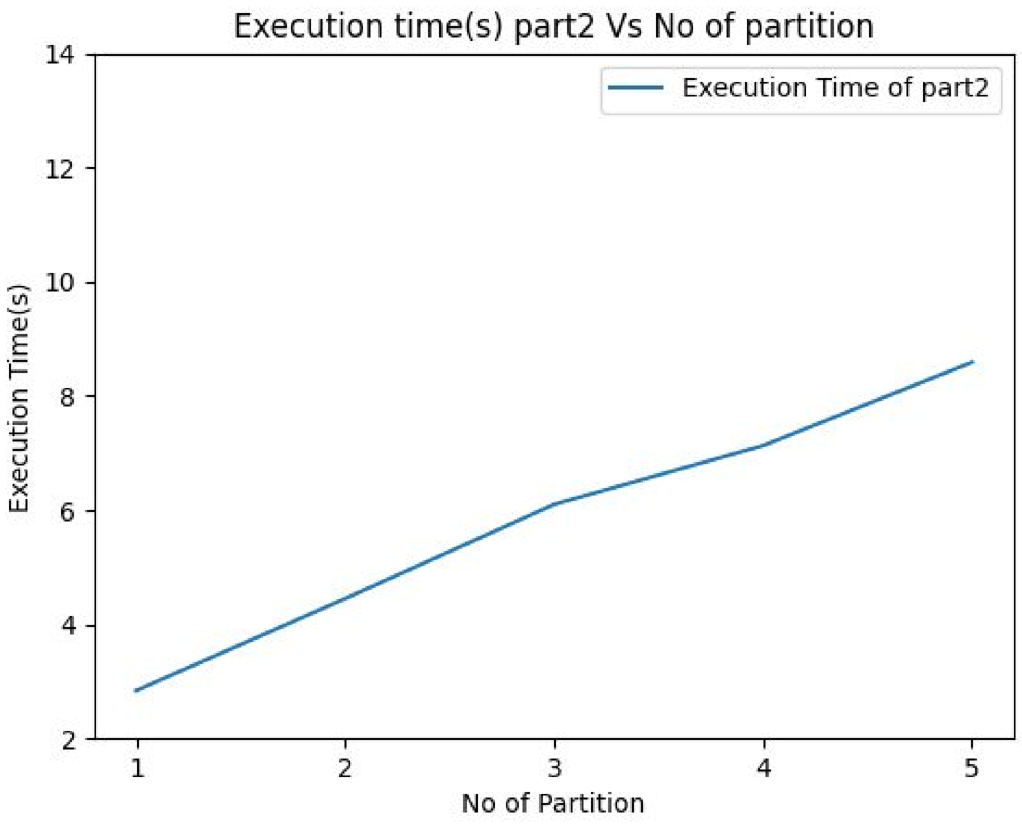
Performance Analysis

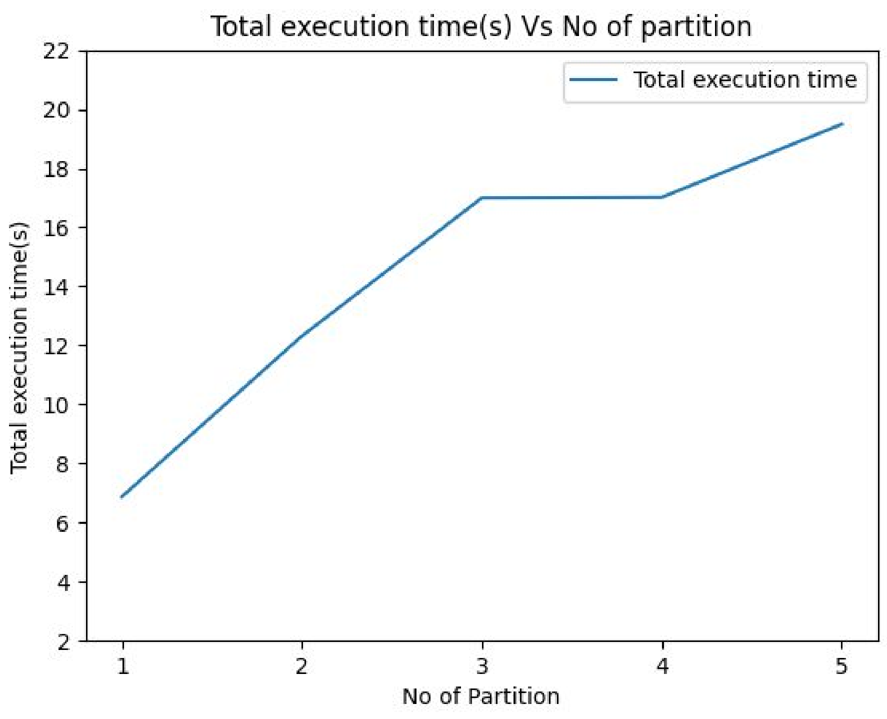
The performance analysis is done based on the number of partitions against the execution time.
Execution time part 1- shows the time taken to execute the first part of the problem statement
Execution time part 2- shows the time taken to execute the second part of the problem statement
Total Execution time- shows the time taken to execute the complete program including the time taken to load the data files

Analysis on small data file

No of partition	Execution Time(s) Part1	Execution Time(s) Part2	Loading time(s)	Total Execution Time(s)
1	3.99	2.84	0.04	6.87
2	7.81	4.45	0.06	12.31
3	10.84	6.10	0.05	16.99
4	9.82	7.13	0.06	17.01
5	10.83	8.59	0.07	19.49

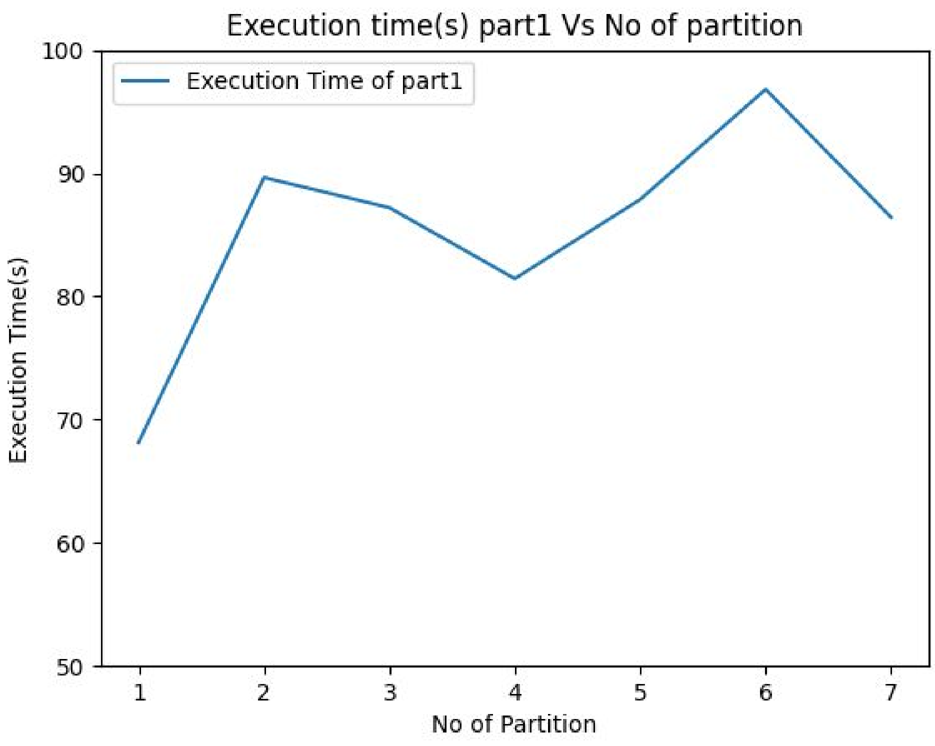

Execution time(s) part1 Vs No of partition
Execution Time of part1
Execution Time(s)
No of Partition
14
12
10
8
6
4
2
1
2
3
4
5

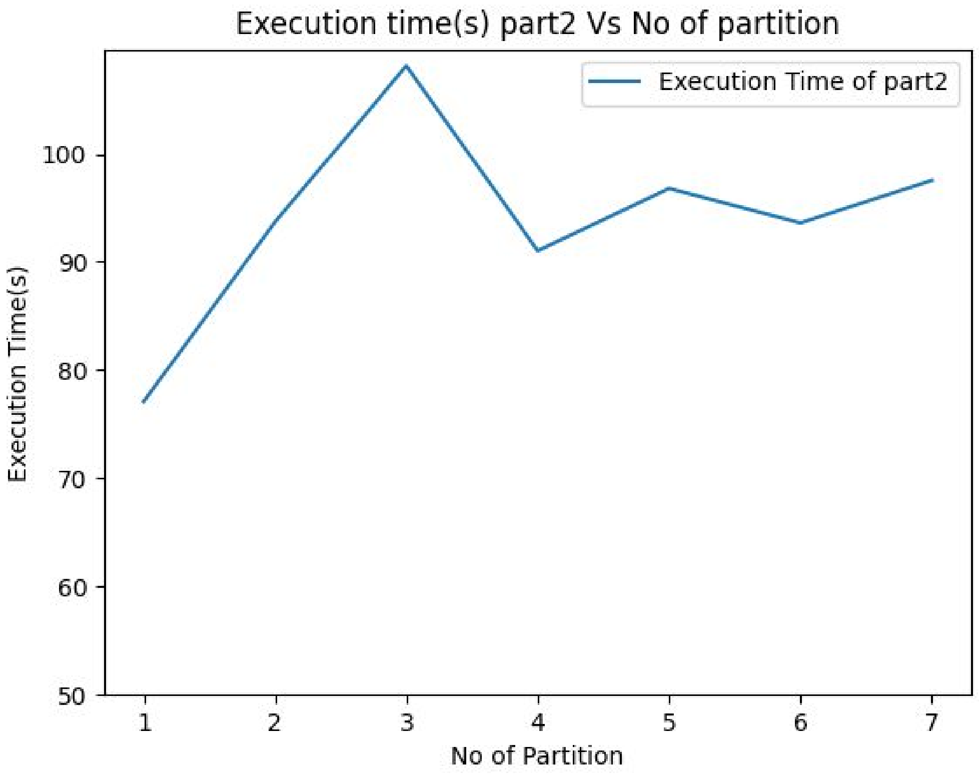

Execution time(s) part2 Vs No of partition
Execution Time of part2
Execution Time(s)
No of Partition
14
12
10
8
6
4
2
1
2
3
4
5

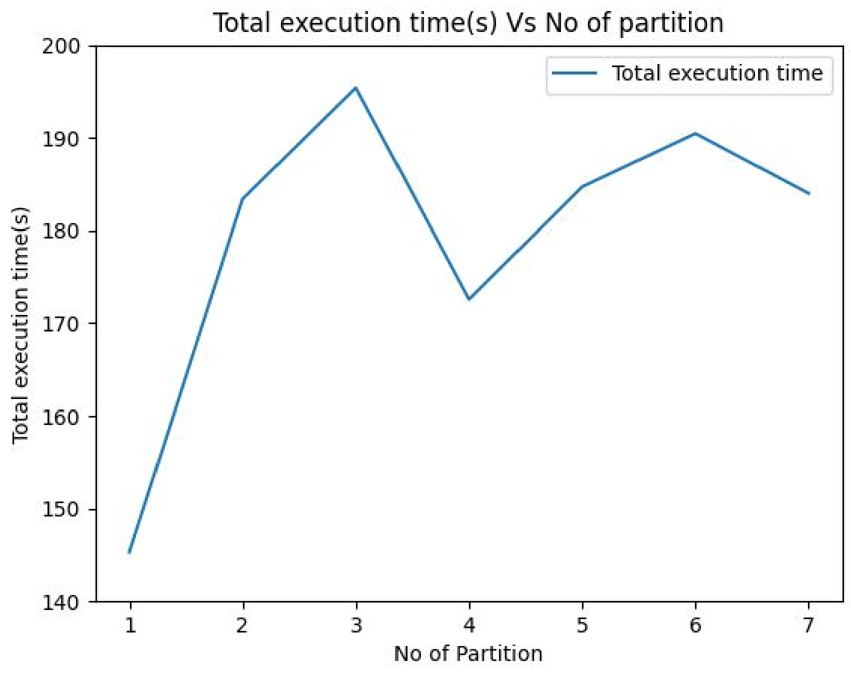


Analysis on large data file

No of partition	Execution Time(s) Part1	Execution Time(s) Part2	Loading time(s)	Total Execution Time(s)
1	68.13	77.09	0.05	145.27
2	89.65	93.69	0.05	183.39
3	87.20	108.12	0.09	195.41
4	81.44	91.04	0.09	172.57
5	87.87	96.79	0.05	184.71
6	96.81	93.61	0.04	190.46
7	86.42	97.53	0.03	183.99


Execution time(s) part1 Vs No of partition
Execution Time of part1
Execution Time(s)
No of Partition
100
90
80
70
60
50
1
2
3
4
5
6
7


Execution time(s) part2 Vs No of partition
Execution Time of part2
Execution Time(s)
No of Partition
100
90
80
70
60
50
1
2
3
4
5
6
7



Analysis

However we have achieved the best execution time when using the default configuration and then using cache

Strategy	Execution Time(s) Part1	Execution Time(s) Part2	Loading time(s)	Total Execution Time(s)
Default configuration	51.98	53.65	0.05	105.68
With Cache	0.90	73.54	0.05	74.49

PART 2

Twitter Sentiment Analysis- Problem Statement

We are going to develop an emerging topic detection from Twitter streaming, and you will do sentiment analysis on the related tweets so that we can see the public opinion on this emerging topic. We need to develop an automatic system that helps us decide the currently emerging topic from the twitter streaming data.

Implementation:

We have implemented this part of the assignment using Spark Streaming, Scala and Stanford CoreNLP. The consumerKey, consumerSecret, accessToken, accessTokenSecret are passed in as the command line argument for completing the required login functionality. Then we create a stream for streaming in the twitter data using the TwitterUtils and then we filter the streamed in tweets for tweets in english so that we will be getting the best results.

Then we extract the tweet text, tag and sentiment using the Stanford CoreNLP and store them in the Dataset and iterate over each RDDS, create a temporary view called 'Sentiments' and make use of the spark sql to show the Top 20 items after processing each window.

We have controlled our log to show only warnings so that the console appears more readable and easy to understand the output of the twitter sentiment analysis program.

We use the annotation pipeline from Stanford coreNLP which is an object used to store an analysis of the piece of text that is processed. Then we extract the sentiments from the text and find the average sentiment by using the following formula:

averageSentiment = sentiments.sum / sentiments.size
If the sentiment of a tweet is empty we assign main sentiment and weighted sentiment to -1.
Then we match the weighted sentiment with its class using the following criteria in the Stanford coreNLP

```
weightedSentiment match {
    case s if s <= 0.0 => NOT_UNDERSTOOD
    case s if s < 1.0 => VERY_NEGATIVE
    case s if s < 2.0 => NEGATIVE
    case s if s < 3.0 => NEUTRAL
    case s if s < 4.0 => POSITIVE
    case s if s < 5.0 => VERY_POSITIVE
    case s if s > 5.0 => NOT_UNDERSTOOD
  }
```

We have tried using sliding window in the code by

```
val hashTags = englishTweets.flatMap(status =>
status.getHashtagEntities.map(_.getText.toLowerCase))
val tagCounts = hashTags.window(Minutes(1),
Seconds(5)).countByValue()
```

We have analyzed the code by varying the batch duration, the things we have observed from our code

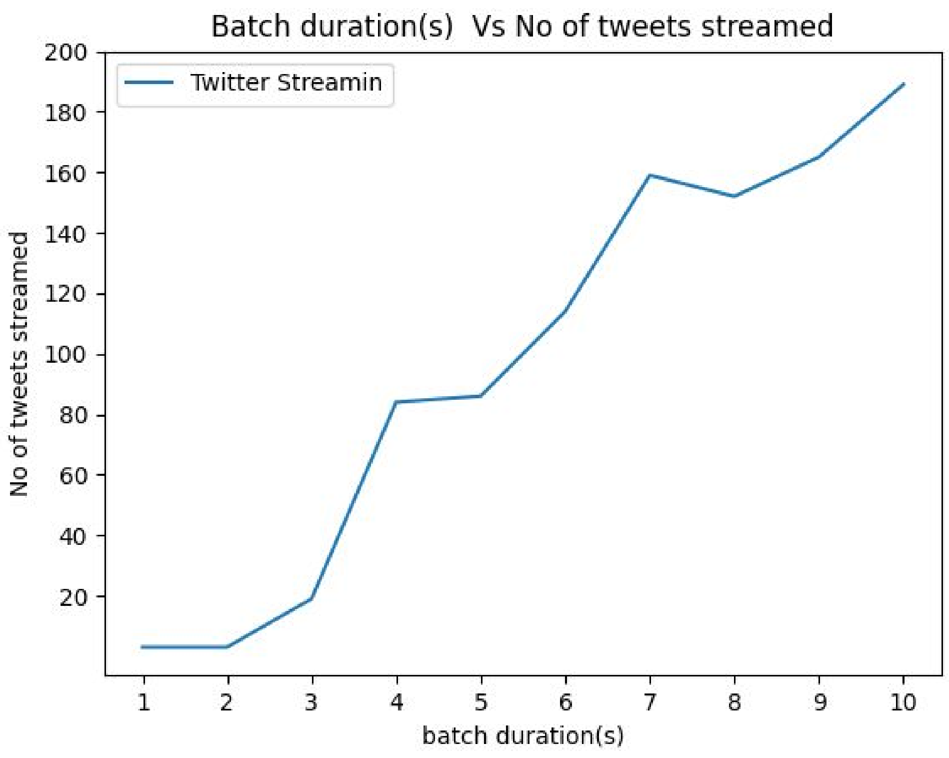
1) when we give a very short batch duration like 1s the no of tweets we get after executing the first batch is low as we are also filtering out tweets in other languages, so we got an output after processing a few batches

2) When we kept the batch duration as 5 or above, we got enough amount data during the first batch to produce an output
3) We are receiving each set of output in intervals that is equal to the batch duration we have specified.

Analysis of Number of tweets that were streamed in vs the batch duration

Batch Duration	No of Tweets
1	3
2	3
3	19
4	84
5	86
6	114
7	159
8	152
9	165
10	189

Since the spark streaming works by dividing the stream into continuous sequences of small RDDs based on the batch duration, the above table shows the batch size parameters and the corresponding number of tweets that were obtained in the first interval of the streaming. As there is continuous streaming of tweets we considered one interval to show the analysis.


Batch duration(s) Vs No of tweets streamed
Twitter Streamin
No of tweets streamed
batch duration(s)
200
180
160
140
120
100
80
60
40
20
1
2
3
4
5
6
7
8
9
10