Real Time Stream Analysis With Spark

Objective: -

The goal of this programming assignment is to analyse data streams over network in real time and derive valuable insights. Two important factors to be focused in this assignment are accuracy of results and how fast were results obtained.

Problem Statement: -

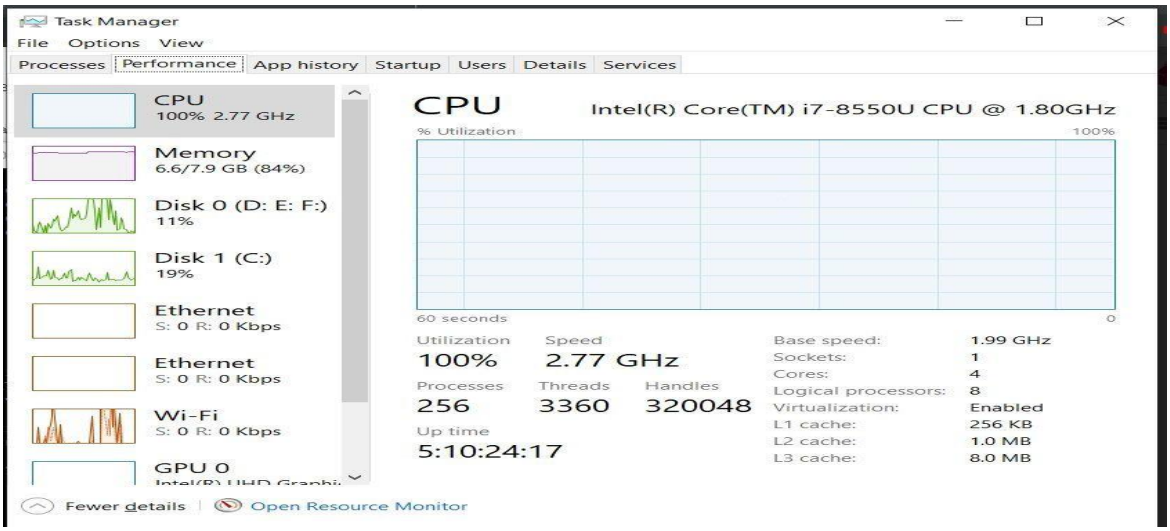
1) Determine the moving average of 'dOctects' field over time window of 1 minute. Along with average also list the largest value corresponding to each one minute window.
2) List out all the "srcaddr" & "dstaddr" that appears twice in a sliding window of 2 minutes.

Input: -

- data-generator.jar will continuously generate UDP packets over the port mentioned. In our case considering rate of transmission as 50000.

System Configuration: -

- Processor: - Intel® Core™ i7-8550HQ CPU @ 1.80GHz
- Installed memory (RAM): - 8.00 GB (7.9 GB)
- System type: - 64-bit Operating System, x64-based processor



Task Manager Snapshot

Setup and execution:-

Requirements: -

- Requires Spark 2.2+.
- Python environment (Canopy), JDK (1.8.0_211 on local system), Spark 2.2+ and winutils(Windows binaries for Hadoop versions)
- StreamingContext is the extension to Spark Core API which will be required to process streaming data.

Spark APIs:-

- Spark SQL: For processing the structured stream.
- SparkStream: Primary API for handling Spark stream.
- Textfilestream: Used for inserting stream data into text files.

Algorithm: -

a) **Part 1**

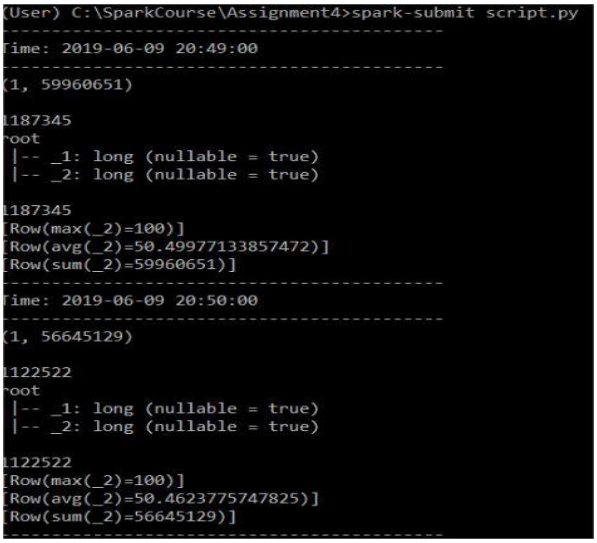
- Running the jar file using the following command:
- java -jar data-generator.jar --destIPaddress 127.0.0.1 --destPortNumber 9876 --transmissionTime 30 --transmissionRate 50000
- This will execute the jar which is responsible for generating packets with random “dOctets”, “srcaddr” and “dstaddr” data fields. Also the timestamp field is of important while processing the packets.
- Jar is transmitting UDP packets at the rate of 50000 packets per second for next 30 minutes on port number 9876.
- Since we have UDP streaming and we know the structure or fields of the incoming stream we will use Structural Stream processing after textstreaming.
- Python script divide the streaming UDP packets into multiple files with cutover interval as 1 minute. Since we are running jar file for 30 minutes, we will have 30 files created containing UDP packet data.
- Now using textfilestream, RDD and pyspark sql, sparkstream will process the individual files simultaneously while it’s been generated and extract needed data and get the average and maximum of “dOctets”.

b) **Part 2**

- It is same as above approach but in this case since it is sliding window, we will generate files at one minute interval and merge the files according to the needed intervals.
- Once we have the files combined, we used simple pyspark sql query two group the “srcaddr” & “dstaddr” fields having count more than 1.
- So for instance during the first and second minute we are getting the '10.194.25.65,10.194.25.69' as the repeating pair of “srcaddr” & “dstaddr”.

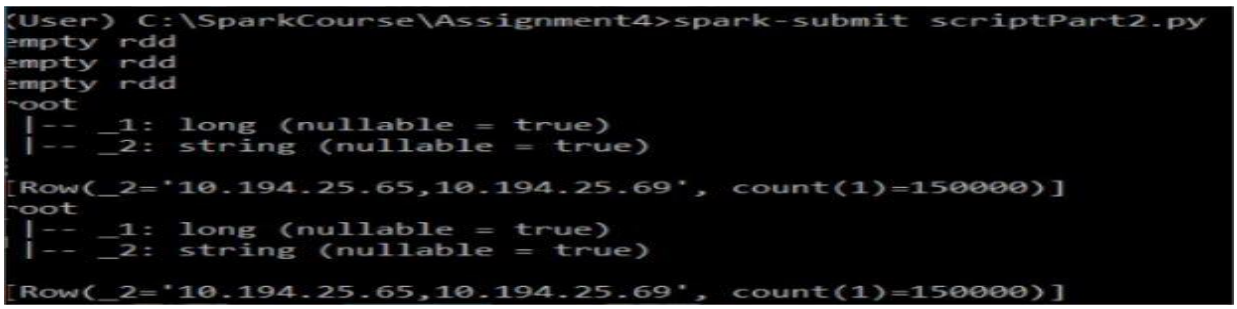
Output:-

1) Below is the output for the first two minutes for the problem number 1. For whole output please check output.txt file.



Output for Part 1

2) For problem number 2 below is the output for 2 sliding window intervals. Complete output is attached with source code.



Output for Part 2

Data Visualization: -

1) Below is the comparison for the packet which is received at the port and the time it takes to finally give the output after 30 minutes.

Event Time v/s Processing Time (In Seconds)			
	First Run	Second Run	Third Run
Event Time	1800	1800	1800
Processing Time	1810	1813	1809

Comparison 1

Performance Tuning:-

- Since it is Spark stream and the performance highly depends on the numbers of packets Spark can process per second, it's was important to consider the cache/processing rate of the packets along with transmission rate.
- So using the Backpressure concept of reactive stream processing systems, we can have optimum transmission rate taking into consideration the cache handling capacity.
- In context of Spark streaming whenever there is overloading of executors they simply notify the upstream to adjust the rate according to the handling capacity of executors.
- We used spark.streaming.backpressure.enabled = true option to make sure data is processed effectively without overloading with rate of transfer as 50K.