Spark Streaming Project Solution

NOTES:

1. I had modified getdata.py to sleep for 50 seconds instead of 45 seconds so I get one file per minute.

2. Please ensure that directory in which python script is saving the output files, should be entered as input directory to Spark program and that directory should not contain any file other than Stock json data files saved by python program.

3. I have included modified getdata.py as well along with other artifacts. Just note that I had used my own key to fetch NYSE data but while submitting, I have changed back the key to what was provided in the file attached in project session 2.

4. Use this command to execute Spark program:
   java -jar SparkApplication.jar <InputPath> <OutputPath>

   e.g. on ec2-instance
   java -jar SparkApplication.jar /home/ec2-user/sparkinput /home/ec2-user/sparkoutput

   e.g. on windows
   java -jar SparkApplication.jar C:\temp\sparkinput C:\temp\sparkoutput

   In both cases above, first put SparkApplication.jar in current directory.

5. I have used Spark Streaming SQL for all 4 analysis.

6. I have compiled the project (SparkStreamingSQL) into a jar which has all dependencies included.

7. I have run the code on 14-02-2019 at 2 different times and have enclosed artifacts for both run.

8. JSON.zip contains 2 folders namely JSON-1 and JSON-2 for json input files set of both run.

9. Output.zip contains 2 folders namely Output-1 and Output-2 for output files set of both run.

10. Output.pdf contains screenshots and logs of both run.

11. SparkApplication.zip contains eclipse project folder namely SparkStreamingSQL.

Main Class:

1. I have created a class (SparkStreamingSQL) in a file namely SparkStreamingSQL.java which is the main class having all the code for streaming.

Main method:

In main() method of the SparkStreamingSQL class which is the main entry point in SparkStreaming:

1. Setup Spark configuration
2. Setup Streaming context with batch interval of 1 minute
3. Setup Spark session
4. Setup SQLContext
5. Setup check pointing
6. Convert input data files into DStream
7. Create different window on DStream for different analysis based on Problem Statement
8. Perform all 4 analysis on respective window stream
9. Display and save output
10. Start, wait for 30 minutes (and 10 seconds) before Termination and Close stream

Analysis - 1: Using Spark Streaming SQL

Simple Moving Average Closing Price for all stocks in the stream window:

1. On original dstream received from textFileStream, use window of 10 minutes with sliding interval as 5 minute and use foreachRDD method.
2. Inside foreachRDD, check if rdd is not empty and then read and parse input rdd in the streaming window using json() method and convert into Dataset data.
3. Select symbol and closing price from the data and create another Dataset data1.
4. Create temporary view on data1 and name it as "simplemovingavg" so SQL like operations can be performed.
5. Using sql() method, in main select clause, select symbol and avg of closing price as avg_closing_price from simplemovingavg group by symbol order by avg_closing_price in descending order and then symbol in ascending order and create another Dataset avgData.
6. **Print "Simple Moving Average Closing Price" along with current timestamp.**
7. Show all records from avgData and this the average closing price for all stocks.
8. Coalesce all rdds into 1 partitions and order by avg_closing_price in descending order and then symbol in ascending order and save as text files in a sub directory under output directory.

Analysis - 2: Using Spark Streaming SQL

Maximum Profitable Stock having highest (average closing price-average opening price) as profit in the stream window:

1. On original dstream received from textFileStream, use window of 10 minutes with sliding interval as 5 minute and use foreachRDD method.
2. Inside foreachRDD, check if rdd is not empty and then read and parse input rdd in the streaming window using json() method and convert into Dataset data.
3. Select symbol, closing price and opening price from the data and create another Dataset data2.
4. Create temporary view on data2 and name it as "maxprofit" so SQL like operations can be performed.
5. Using sql() method, in from clause (select symbol, avg of closing price as avg_close and avg of opening price as avg_open from maxprofit group by symbol) and then in main select clause, select symbol, avg_close, avg_open and avg_close-avg_open as profit order by profit in descending order and then symbol in ascending order and create another Dataset profitData.
6. **Print "Maximum Profitable Stock" along with current timestamp.**
7. Show top first record from profitData and this the stock having maximum profit
8. Coalesce all rdds into 1 partitions and order by profit in descending order and then symbol in ascending order and save as text files in a sub directory under output directory.

Analysis - 3: Using Spark Streaming SQL

Relative Strength Index (RSI) for each stock in the stream window:

1. On original dstream received from textFileStream, use window of 10 minutes with sliding interval as 1 minute and use foreachRDD method.
2. Inside foreachRDD, check if count >= 10 and then read and parse input rdd in the streaming window using json() method and convert into Dataset data.
3. Select symbol, timestamp and closing price from the data and create another Dataset data3.
4. Create temporary view on data3 and name it as "strengthindex" so SQL like operations can be performed.
5. Using sql() method, in with clause, (first select * and row_number() partition by symbol, order by timestamp so record with latest timestamp appears in last) and then in main select clause, (compute gain and loss based on closing price of previous record and current record by joining on data of previous record and current record and create another Dataset gainLossData.
6. Create temporary view on gainLossData and name it as "gainlossindex" so SQL like operations can be performed.
7. Using sql() method, select symbol, average gain and average loss from gainlossindex group by symbol and create another Dataset avgGainLossData.
8. Create temporary view on avgGainLossData and name it as "strengthindex2" so SQL like operations can be performed.
9. Using sql() method, select symbol and compute rsi:
   (a) If avg gain is 0 then rsi is 0
   (b) If avg loss is 0 then rsi is 100
   (c) Otherwise compute rsi using the formula mentioned in the problem statement for rsi and order by rsi in descending order and then symbol in ascending order and create another Dataset rsiData.
10. **Print "Relative Strength Index" along with current timestamp.**
11. Show data from rsiData.
12. Coalesce all rdds into 1 partitions and order by rsi in descending order and then symbol in ascending order and save as text files in a sub directory under output directory.

Analysis - 4: Using Spark Streaming SQL

Highest Trading Volume stock in the stream window to buy:

1. On original dstream received from textFileStream, use window of 10 minutes with sliding interval as 10 minute and use foreachRDD method.
2. Inside foreachRDD, check if rdd is not empty and then read and parse input rdd in the streaming window using json() method and convert into Dataset data.
3. Select symbol and volume from the data and create another Dataset data4.
4. Create temporary view on data4 and name it as "maxvolume" so SQL like operations can be performed.
5. Using sql() method, in main select clause, select symbol and sum of volume as total_volume from maxvolume group by symbol order by total_volume in descending order and then symbol in ascending order and create another Dataset volumeData.
6. **Print "Buy this Stock having Highest Volume Traded" along with current timestamp.**
7. Show top first record from volumeData and this the stock to buy.
8. Coalesce all rdds into 1 partitions and order by total_volume in descending order and then symbol in ascending order and save as text files in a sub directory under output directory.

END OF DOCUMENT