JIGSAW ACADEMY
THE ONLINE SCHOOL OF ANALYTICS

Case Study: Spark Streaming – Practice (Non-Graded) Assignment

DStream and RDD Operations

Business Scenario:

We have NYSE stock data set. One file has the maximum stock price of the previous day consisting of stock symbol of the company and the previous max price. The other is streaming data each record of which consists of the stock symbol, time stamp in HH:MM:SS format and the current stock price.

Data Set and Data Dictionary:

The data fields and the sample records are as shown below.

Previous Max Price file is a comma-separated (csv) file without header having records in the following format.

Stock Symbol of Company	Previous Max Price
JAH,	60.46
JAS,	38.06
JBI,	28.7
JBI,	28.7
JBJ,	26.8

Streaming data contains records in the following format with each field delimited by a comma.

Stock Symbol of Company	Time Stamp	Current Stock Price
JTA,	08:21:37,	25.15
JGT,	08:21:38,	18.25
JHI,	08:21:39,	20.21
JKH,	08:21:40,	100.56
JGT,	08:21:41,	15.53

Problem Statement:

Write a PySpark streaming application which:

a) Reads the static data of previous max price
b) Captures the streaming data every few seconds (it can be as small interval as 2 or it can be 10 seconds as per your choice)
c) Compares the current stock price of each company from the data stream with the static data of previous max price and
d) Flags i.e. displays the details when the current price reaches or goes above the previous max price.

Solution:

The solution is provided in the iPython notebook file. The steps we took broadly are as follows.

1. Create a `SparkContext` with two execution threads, and `StreamingContext` with batch interval of 1 second.

2. Read and cache previous max price into an RDD. Each record of the RDD is a line of text with fields delimited by comma. We need to split the text line at the delimiter into individual strings of stock symbol and previous max price. Convert the previous max stock price to float type.
3. Using spark streaming context, we can create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. `localhost`) and port (e.g. `9999`). This DStream represents the stream of data.
4. Each record in this DStream is a line of text with fields delimited by comma. We will split the test line at the delimiter into individual strings. Convert each record into record to key-value pair with key being the stock symbol and value being a tuple of time stamp string and current price of float type.
5. We need to join data stream records with those of the previous max RDD on stock symbol field. We can use the `join` function, but since it is an RDD-to-RDD function we need to use `transform` method on the stream so that we can apply it. For further documentation see: `https://spark.apache.org/docs/2.1.1/streaming-programming-guide.html#transformations-on-dstreams`
6. Joined records will be of the form key-value: `stock_symbol, ((timestamp,current price),previous max price)`. So, `record[0]` will be `stock_symbol`, `record[1]` will have 2 fields. In the two fields of `record[1]` - one is `record[1][0]` which has `(time spamp, current price)`; the other is `record[1][1]` which has previous max price.
7. We need to filter for those records in which current price is greater than or equal to previous max price.
8. Then we will use pprint (pretty print) function to print record matching the filter condition.

You can run Jupyter notebook from one terminal from a working directory where you have all the required files which are:

a) The iPython notebook file
b) Shell script that simulates socket data stream from a port number 9999
c) Input data files for for stock stream and previous max price

To simulate the continuous stream of records we will use the shell script `stock_details_socket.sh` and the input data file `stock_stream_data.csv` as below from the $ prompt of another terminal.

```
$ ./stock_details_socket.sh stock_stream_data.csv | nc -lk 9999
```

The above command takes one line from the input data file and sends it to the port 9999 every half-a-second. This is the port we used in the code from which we will read data streaming in.

You can stop both the jobs by pressing Control+C from respective terminals.

You can also run the program from its `.py` file `using spark-submit` command as well.

Test File Stream

The above case study illustrated `socketTestStream` which creates a DStream from text data received over a TCP socket connection. Besides sockets, the StreamingContext API provides methods for creating DStreams from files as input sources also. PySpark (Python API of Spark) supports textFileStream.

A DStream can be created by passing a directory name to textFileStream. Spark Streaming will monitor the given directory and process any files created in that directory (files written in nested directories not supported). Note that:

- The files must have the same data format.
- The files must be created in the directory by atomically moving them into the data directory or renaming them.
- Once moved, the files must not be changed. If the files are being continuously appended, the new data will not be read. Any new files created will be read.

The only change we need to do in the above code is at step #3 where we need to specify `testFileStream` by passing a data directory name as the parameter.

The accompanying shell script `stock_details_textfile.sh` can be run as follows .

```
$ ./stock_details_textfile.sh stock_stream_data.csv 100
```

It will create new files every half-a-second in the directory `/tmp/streamdir` with file names as `file_100`, `file_101` etc.

Each of these files contains one line from the stream input data. Our program picks up these files every one second and processes the content of the files in the same way as in the socket example.