JIGSAW ACADEMY
THE ONLINE SCHOOL OF ANALYTICS

Spark Streaming – Practice (Non-Graded) Assignment

Spark Streaming with Kafka as data source
Direct (non-receiver based) Approach

Introduction:
Spark streaming can process data from simple sources like a file or socket stream as well as advanced sources such as Kafka using additional utility classes. We can develop spark streaming applications to process messages of a Kafka topic. This use case explains how word count can be done on the messages of a Kafka topic. As and when a message is available on a Kafka topic our application will consume it and performs word count on the individual words of the message. In this we will use what is called *Direct* approach. Unlike the previous approach this does not use receivers to get the data from Kafka.

Kafka in a Nutshell:
Kafka is a distributed streaming platform that runs on a cluster and facilitates:
a) Publish and subscribe to streams of messages
b) Stores streams of messages or records in a durable manner for pre-defined period
c) Processing of messages as they occur in real-time

Messages can be published to Kafka *topics* by a producer. Kafka stores the stream of messages for a specified retention period in the cluster. The stream is partitioned and the partitions are distributed across Kafka cluster on the servers or *brokers* as they are called. The messages can be consumed and processed by *consumers* in real-time as and when they are published or within the retention period as per the processing requirement.

Kafka uses Zookeeper to monitor whether the brokers are live and available.

This use case explains how we can write Spark streaming applications that can consume and process streaming data in real-time from topics of Kafka clusters. Our Spark streaming application performs a word count on the messages published in a Kafka topic and displays each word & the number of times it occurred in that message in real-time. So the word count gets done and shows up as and when the message is published to the topic.

Running the Kafka - Spark Streaming Application:
We first need to run Zookeeper and then Kafka. Next we need to create a topic and then start sending messages on to the topic using Kafka console producer. We can also test the producer by running a Kafka console consumer and checking if it can get the messages and display on console or not.

The following set of commands performs the above tasks.

Step-0: We may need an initial clean up on Jigsaw VM, as sometimes a zookeeper instance might be running in the background due to some residual effect. So we can stop the unwanted zookeeper service running the following command on a terminal say Terminal #1.

```
sudo /usr/local/kafka/bin/zookeeper-server-stop.sh
/usr/local/kafka/config/zookeeper.properties
```

Step-1: We can start a fresh instance of zookeeper and run it in the background with the following command.

```
zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties >
/tmp/zkservinit.log 2>&1 &
```

Step-2: Now we will start a Kafka server instance and run it in the background with the following command.

```
kafka-server-start.sh $KAFKA_HOME/config/server.properties >
/tmp/kfservinit.log 2>&1 &
```

Step-3: The following command lists the existing topics.

```
kafka-topics.sh --list --zookeeper localhost:2181
```

Step-4: If we don't find the topic named *kafka-streaming1* in the list we can create it with the command below.

```
kafka-topics.sh --create --zookeeper localhost:2181 --replication-
factor 1 --partitions 1 --topic kafka-streaming2
```

Step-5: Check the if the topic is created or not by listing the topics again with the list command.

```
kafka-topics.sh --list --zookeeper localhost:2181
```

Step-6: The command below gives the description of our topic.

```
kafka-topics.sh --zookeeper localhost:2181 --describe --topic kafka-
streaming2
```

Step-7: We can now start producing messages and send them on to out topic using Kafka console producer using the following command and typing in any test lines of our choice.

```
kafka-console-producer.sh --broker-list localhost:9092 --topic kafka-
streaming2
>spark streaming with kafka streaming data on kafka with spark
```

Step-8: On another terminal, Terminal #2, using a Kafka console consumer as shown below, we can test if the messages are being sent by the producer to the topic properly. The Kafka console consumer should get the messages typed into the producer in the above step and display them on console.

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
kafka-streaming2 --from-beginning
```

This is an optional step. It can be performed as a verification step.

Step-9: On a separate terminal say, Terminal #3, we can run our Kafka-Spark-Streaming application as shown below. The code is in the file named *streamingkafka2.py*

```
spark-submit --master local[2] --packages org.apache.spark:spark-
streaming-kafka-0-8_2.11:2.0.2 streamingkafka2.py
```

The option *--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.0.2* adds the required libraries and the dependencies while running the application using *spark-submit*

Now on terminal #1 if we enter new lines as shown in example below, we can see on terminal #3 the words from these messages are counted in real-time and the word count gets displayed.

```
>test kafka and spark streaming with streaming on kafka and spark
```

To stop the streaming application we can press Control+C. Also to stop Kafka console consumer running on terminal #2 also we can press Control+C. Similarly to stop Kafka console produce as well we can press Control+C.

Step-10: Remember to delete the topic with the command below as a final clean up at the end i.e. after running our Kafka-Spark Streaming application successfully.

```
kafka-topics.sh --zookeeper localhost:2181 --delete --topic kafka-
streaming2
```

Application Code:

To walk through the Kafka spark streaming application code:

- After importing the required pyspark packages, we create SparkContext as sc with two threads and then SparkStreamingContext ssc with batch interval of 1 second.
- We then import KafkaUtils and create an input DStream using *KafkaUtils.createDirectStream* passing the parameters – Spark Streaming Context, topic and broker(s).
- This is called *Direct* approach as it does not involve any receivers unlike the previous approach. This approach ensures stronger end-to-end guarantees. Instead of using receivers to receive data, this approach periodically queries Kafka to get the latest offsets in each topic+partition. Accordingly it defines the offset ranges to process in each batch of messages. When the jobs to process the data are launched, Kafka's simple consumer API is used to read the defined ranges of offsets from Kafka (similar to read files from a file system).
- The data is then handled as a normal RDD in our application to perform word count.

The code is given as a Jupyter Notebook file *streamingkafka2.ipynb.* The first line in this code specifies the required packages and dependencies to be added at the time of running the application similar to the command line option we used while running the application *streamingkafka2.py.*