K-CS259: Data Analysis Using Apache Kafka

Riti Gupta
Computer Science Department San Jose State University San Jose, CA 95192 408-924-1000
riti.gupta@sjsu.edu

APPLICATION 1: Log Analysis

Producer.java

```
import org.apache.kafka.clients.producer.*;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Properties;
import java.util.Random;
import java.util.concurrent.ExecutionException;

class Producer {

    public static void main(String[] args) throws ExecutionException,
InterruptedException {
        String server = "127.0.0.1:9092";
        String topic = "user_data";

        Producer producer = new Producer(server);
        Long i = Long.valueOf(0);
        while(true) {
            String user = "user" + i;
            i++;
            String info = "";
            Random r = new Random();
            int num = r.nextInt(3);
            if(num == 0) {
                info = "INFO";
            } else if (num == 1) {
                info = "WARNING";
            } else {
                info = "ERROR";
            }
            producer.put(topic, user, info);
        }
    }

    // Variables

    private final KafkaProducer<String, String> mProducer;
    private final Logger Logger = LoggerFactory.getLogger(Producer.class);

    // Constructors

    Producer(String bootstrapServer) {
        Properties props = producerProps(bootstrapServer);
        mProducer = new KafkaProducer<>(props);
```

```
        Logger.info("Producer initialized");
    }

    // Public
    void put(String topic, String key, String value) throws ExecutionException,
InterruptedException {
        Logger.info("Put value: " + value + ", for key: " + key);

        ProducerRecord<String, String> record = new ProducerRecord<>(topic, key,
value);
        mProducer.send(record, (recordMetadata, e) -> {
            if (e != null) {
                Logger.error("Error while producing", e);
                return;
            }

            Logger.info("Received new meta. Topic: " + recordMetadata.topic()
                    + "; Partition: " + recordMetadata.partition()
                    + "; Offset: " + recordMetadata.offset()
                    + "; Timestamp: " + recordMetadata.timestamp());
        }).get();
    }

    void close() {
        Logger.info("Closing producer's connection");
        mProducer.close();
    }

    // Private

    private Properties producerProps(String bootstrapServer) {
        String serializer = StringSerializer.class.getName();
        Properties props = new Properties();
        props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServer);
        props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, serializer);
        props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, serializer);

        return props;
    }
}
```

Consumer.java

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.spark.streaming.Durations;
import java.util.*;
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.api.java.*;
import org.apache.spark.streaming.kafka010.*;
import scala.Tuple2;
import java.util.regex.Pattern;

class ConsumerSpark {

    private static final Pattern SPACE = Pattern.compile(" ");

    public static void main(String[] args) throws InterruptedException {
        String server = "127.0.0.1:9092";
        String groupId = "application";
        String topic = "user_data";
```

```
        SparkConf sparkConf = new
SparkConf().setAppName("JavaDirectKafkaWordCount").setMaster("local");
        JavaStreamingContext jssc = new JavaStreamingContext(sparkConf,
Durations.seconds(5));
        Set<String> topicsSet = new HashSet<>(Arrays.asList(topic));
        Map<String, Object> kafkaParam = new HashMap<>();
        kafkaParam.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, server);
        kafkaParam.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        kafkaParam.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        kafkaParam.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        // Create direct kafka stream with brokers and topics
        JavaInputDStream<ConsumerRecord<String, String>> messages =
KafkaUtils.createDirectStream(
                jssc,
                LocationStrategies.PreferConsistent(),
                ConsumerStrategies.Subscribe(topicsSet, kafkaParam));
        // Get the lines, split them into words, count the words and print
        JavaDStream<String> lines = messages.map(ConsumerRecord::value);
        JavaDStream<String> words = lines.flatMap(x ->
Arrays.asList(SPACE.split(x)).iterator());
        JavaPairDStream<String, Integer> wordCounts = words.mapToPair(s -> new
Tuple2<>(s, 1))
                .reduceByKey((i1, i2) -> i1 + i2);
        wordCounts.print();
        // Start the computation
        jssc.start();
        jssc.awaitTermination();
    }
}
```

APPLICATION 2: Twitter Data Stream Analysis

Producer.java

```
import org.apache.kafka.clients.producer.*;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Properties;
import java.util.concurrent.ExecutionException;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.concurrent.LinkedBlockingQueue;

import twitter4j.*;
import twitter4j.conf.*;

import twitter4j.StallWarning;
import twitter4j.Status;
import twitter4j.StatusDeletionNotice;
import twitter4j.StatusListener;
import twitter4j.TwitterStream;
import twitter4j.TwitterStreamFactory;
import twitter4j.conf.ConfigurationBuilder;
import twitter4j.json.DataObjectFactory;

class Producer {
```

```
    public static void main(String[] args) throws ExecutionException,
InterruptedException {
        final LinkedBlockingQueue<Status> queue = new
LinkedBlockingQueue<Status>(1000);
        String consumerKey = "FAs7QhvUKsP6n4ZOijUuEAsG7";
        String consumerSecret = "jvimUIrIsAkYHXGBdpJRjvR9ZULayYzuAb4yUMun90sW4fPP92";
        String accessToken = "67322860-oYabBt3LVe48eM1vIMwnJgTmu06RCpjt955EDDFM6";
        String accessTokenSecret = "mmX4fjLY2sBIHwu6sCNXaMwvFYwl5iIn0bvlou4mn6t3u";
        String topicName = "user_data";
        String[] keyWords = {"linux", "ubuntu", "windows", "unix"};
        ConfigurationBuilder cb = new ConfigurationBuilder();

cb.setDebugEnabled(true).setOAuthConsumerKey(consumerKey).setOAuthConsumerSecret(consu
merSecret)

.setOAuthAccessToken(accessToken).setOAuthAccessTokenSecret(accessTokenSecret);
        // Create twitterstream using the configuration
        TwitterStream twitterStream = new
TwitterStreamFactory(cb.build()).getInstance();
        StatusListener listener = new StatusListener();
        twitterStream.addListener(listener);

        // Filter keywords
        FilterQuery query = new FilterQuery().track(keyWords);
        twitterStream.filter(query);

        String server = "127.0.0.1:9092";

        Producer producer = new Producer(server);
        int i = 0;
        int j = 0;

        // poll for new tweets in the queue. If new tweets are added, send them
        // to the topic
        while (true) {
            Status ret = queue.poll();

            if (ret == null) {
                Thread.sleep(100);
            } else {
                for (HashtagEntity hashtage : ret.getHashtagEntities()) {
                    System.out.println("Tweet:" + ret);
                    System.out.println("Hashtag: " + hashtage.getText());
                    producer.put(topicName, Integer.toString(i++), ret.getText());
                }
            }
        }
    }
    private final KafkaProducer<String, String> mProducer;
    private final Logger Logger = LoggerFactory.getLogger(Producer.class);

    // Constructors

    Producer(String bootstrapServer) {
        Properties props = producerProps(bootstrapServer);
        mProducer = new KafkaProducer<>(props);
```

```
        Logger.info("Producer initialized");
    }

    // Public

    void put(String topic, String key, String value) throws ExecutionException,
InterruptedException {
        Logger.info("Put value: " + value + ", for key: " + key);

        ProducerRecord<String, String> record = new ProducerRecord<>(topic, key,
value);
        mProducer.send(record, (recordMetadata, e) -> {
            if (e != null) {
                Logger.error("Error while producing", e);
                return;
            }

            Logger.info("Received new meta. Topic: " + recordMetadata.topic()
                    + "; Partition: " + recordMetadata.partition()
                    + "; Offset: " + recordMetadata.offset()
                    + "; Timestamp: " + recordMetadata.timestamp());
        }).get();
    }

    void close() {
        mLogger.info("Closing producer's connection");
        mProducer.close();
    }

    // Private

    private Properties producerProps(String bootstrapServer) {
        String serializer = StringSerializer.class.getName();
        Properties props = new Properties();
        props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServer);
        props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, serializer);
        props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, serializer);

        return props;
    }
}
```

Consumer.java

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.spark.streaming.Durations;
import java.util.*;
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.api.java.*;
import org.apache.spark.streaming.kafka010.*;
import scala.Tuple2;
import java.util.regex.Pattern;

class Consumer2 {

    private static final Pattern SPACE = Pattern.compile(" ");

    public static void main(String[] args) throws InterruptedException {
        String server = "127.0.0.1:9092";
        String groupId = " application";
        String topic = "user_data";
```

```
        SparkConf sparkConf = new
SparkConf().setAppName("JavaDirectTwitter").setMaster("local");
        JavaStreamingContext jssc = new JavaStreamingContext(sparkConf,
Durations.seconds(5));
        Set<String> topicsSet = new HashSet<>(Arrays.asList(topic));
        Map<String, Object> kafkaParam = new HashMap<>();
        kafkaParam.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, server);
        kafkaParam.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        kafkaParam.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        kafkaParam.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        JavaInputDStream<ConsumerRecord<String, String>> messages =
KafkaUtils.createDirectStream(
                jssc,
                LocationStrategies.PreferConsistent(),
                ConsumerStrategies.Subscribe(topicsSet, kafkaParam));
        // Get the lines, split them into words, count the words and print
        JavaDStream<String> lines = messages.map(ConsumerRecord::value);
        JavaDStream<String> words = lines.flatMap(x ->
Arrays.asList(SPACE.split(x)).iterator());
        JavaDStream<String> hashTags = words.filter(name -> name.startsWith("#"));
        JavaPairDStream<String, Integer> wordCounts = hashTags.mapToPair(s -> new
Tuple2<>(s, 1))
                .reduceByKey((i1, i2) -> i1 + i2);
        wordCounts.print();
        // Start the computation
        jssc.start();
        jssc.awaitTermination();
    }
}
```