CSC8101 Big Data Analytics
Assignment 2 Specification

February 2016

1 Introduction

This assignment builds on the experience you gained with Spark in the previous assignment regarding document indexing. However, instead of static data, this assignment deals with real-time streaming data arriving from a message broker, in this case Apache Kafka.

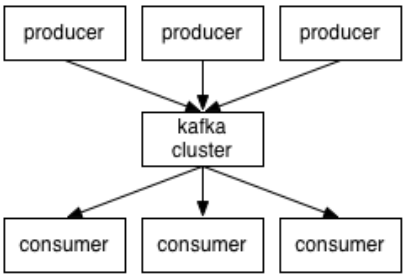
1.1 Spark Streaming

Spark's Streaming module allows you to apply the power of Spark's distributed and resilient processing system to real time data streams. Spark Streaming is a layer on top of Spark's core libraries and allows incoming data to be grouped into small batches, each of which is treated as an individual RDD.

It should be noted that the stream of RDDs (refered to as a DStream) has a different set of available operations to that of standard RDDs. It is **strongly** recommended that your read the Spark Streaming Programming Guide on the Spark website to understand the difference from the core Spark libraries and the constraints that stream processing introduces. The programming guide has examples in several languages (Scala, Java and Python) and details all the available operations for DStreams. Additional language specific implementation details can be found in the appropriate API documentation.

1.2 Kafka

Apache Kafka is a distributed, partitioned, replicated commit log service. It has many uses, however for this assignment we will be using it as a message broker that can store incoming messages from a variety of sources (often called producers) and serve these to multiple consumers.



Message brokers are often employed to serve as an intermediary between different the different layers of a software application stack. They can simplify large distributed computing systems by separating different layers from one another. For example, say you had a large number of sensors deployed around a city, that are reporting to a a cluster of computers that process the sensor information. If in the future you change the location of the cluster, you would have to inform all the sensors (potentially many hundreds of them) of the change of receiving address. Similarly, if

you switch your cluster to a new software system or communication protocol, you would also have to update all of your sensors to use this new system. Placing a message broker between the sensors and the processing infrastructure solves these two issues. The sensors always send to the broker and do not need to be informed of the processing system or its protocols and visa versa.

Message brokers also act as buffers to smooth out spikes in traffic and are often employed in big data processing systems for this purpose. Typically, brokers are deployed on a cluster of machines in the cloud, however for this assignment you will be running a Kafka instance locally on your VM.

Knowing the internal operations of Kafka are not required for this assignment, however reading the introduction on the Kafka website will give you an understanding of the terms used in the setup scripts supplied on your VM and may help in understanding error messages produced by your Spark code in relation to Kafka.

2 Assignment Context

This assignment is based on simulated traffic flow data across 100 sites in a city. The Kafka broker will store messages containing a site identification number, a vehicle licence plate string and a timestamp. This will be supplied as a JSON formatted string. An example message string is given below:

```
{"SiteID":1, "Licence":"A123", "Time": "2016/01/29 15:49:15.158648"}
```

The data for each message is actually loaded from a file on your VM and inputed into your local Kafka instance in real-time.

3 Assessment

3.1 Questions

For this assignment you are required to develop Spark programs to answer the following queries:

3.1.1 Distinct Licences

Query Calculate the number of distinct vehicles, per consecutive minute, through the entire road network (over all 100 sites).

Details Your program should produce a DStream, where each RDD in the DStream contains the number of **distinct** lincences for each minute of the incoming stream. Typically this information would then be saved to a database or forwarded on for further processing. In the case of this assignment you should save the counts for each RDD in the DStream to a file on your VM.

Deliverables

- The source code file for your Spark Streaming program.
- A summary of 1 hour of streaming results. If the query above saves each minute of data to a folder, you should have 60 count values after running the program for 1 hour. You should summaries these in a single text or csv file, where each minute of the hour is clearly linked to its count value. You can create this summary file manually or write an automate script to do it for you. There are no extra marks for automating this summary.

3.1.2 Congested Sites

Query Record all congestion sites within the road network per consecutive minute. A site is defined as congested if there are 200 or more events in a 1 min period.

Details Your program should produce a Dstream, where each RDD only contains SiteID's that have an event count above 200 for the corresponding minute. Typically this information would then be saved to a database or forwarded on for further processing. In the case of this assignment you should save the SiteIDs in the DStream to a file on your VM.

Deliverables

- The source code file for your Spark Streaming program.
- A summary of 1 hour of streaming results. If the query above saves each minute of data to a folder, after running the program for 1 hour you should have 60 files containing the SiteIDs of congested site. You should summaries these in a single text file or csv where each minute of the hour is clearly linked to a list of SiteIDs. You can create this summary file manually or write an automate script to do it for you. There are no extra marks for automating this summary.

3.2 Submission

Each of the required deliverables listed for the queries above should by placed in their own folder within a single zip or tar file. For example, you may have a zip file called `CSC8101-Streaming-MyName-StudentNo.zip`. Within this zip file would be a folder for each query; `Distinct-Licences`, `Congested-Sites` etc which would contain the deliverables listed above.

Do Not simply place all of the raw output from your program into your submission, each query must be clearly separated with source code and summary files in their own folder.

3.3 Mark Scheme

1. Distinct Licences [40]
   - Program [30]
   - Summary Output [10]
2. Congested Sites [60]
   - Program [40]
   - Summary Output [20]

4 Implementation information

As an absolute minimum you should consult the Quick Example code in your chosen language, as this highlights several key differences between how standard Spark programs work and how Spark Streaming programs work. Take particular note of the need to call the start method on the Spark streaming context object in order for the program to begin processing the stream.

For this assignment you will be running Spark locally on your VM as you did in assignment 1. When running Spark Streaming in local mode you should supply `local[2]` to the Spark Context constructor or the `--master` flag of the REPL commands (such as `pyspark`). This is important as Spark Streaming requires a minimum of 2 worker processes to function correctly (see the Spark Streaming Programming Guide for more details).

You program to process the incoming stream of messages and provide answers to the questions in section 3 should be a self contained program in your chosen language (Scala, Java or Python) and not a simple list of copied commands you typed into one of the Spark REPLs (such as `pyspark`.

4.1 Connecting to Kafka

It is recommended that your read the Spark + Kafka integration guide in order to understand how these two systems integrate. There are two methods to connect to Kafka listed on the integration guide. It is recommended that you use Approach 2 (direct connection with no reciever) as Approach 1 is to be deprecated in future versions of Spark.

Section 4.4.1 describes how to start your local Kafka server. The server will be made available on localhost port 9092 (`"localhost:9092"`). This should be used as the `"metadata.broker.list"` key in the Approach 2 connection instructions.

4.1.1 Python

An example of the code to form a Kafka connection in Python is given below:

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

# Create a local StreamingContext
sc = SparkContext("local[2]", "DistinctLicences")

#Set the batch interval in seconds
batch_interval = 10

#Create the streaming context object
ssc = StreamingContext(sc, batch_interval)

#Create the kakfa connection object
KafkaStream = KafkaUtils.createDirectStream(ssc, ["licence"], {"metadata.
    broker.list": "localhost:9092"})
```

4.1.2 Java

An example of the code to form a Kafka connection in Java is given below:

```
import java.util.HashMap;
import java.util.HashSet;

import org.apache.spark.SparkConf;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaPairInputDStream;
import org.apache.spark.streaming.api.java.JavaPairReceiverInputDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.kafka.KafkaUtils;

import kafka.serializer.StringDecoder;

public class DistinctLicences {

  public static void main(String[] args) {

    // Create a local StreamingContext with two working thread and batch
    // interval of 10 seconds
    SparkConf conf = new SparkConf()
                     .setMaster("local[2]")
                     .setAppName("DistinctLicences");

    JavaStreamingContext jssc =
      new JavaStreamingContext(conf, Durations.seconds(10));

    // Create the kafka connection settings
    HashMap<String, String> kafkaParams = new HashMap<String, String>();
    kafkaParams.put("metadata.broker.list", "localhost:9092");

    // Set the topic to connect to on the broker
    HashSet<String> topics = new HashSet<String>();
    topics.add("licence");

    // Create the DStream of raw JSON messages from the broker
    JavaPairInputDStream<String, String> directKafkaStream =
      KafkaUtils.createDirectStream(jssc,
                                    String.class,
                                    String.class,
                                    StringDecoder.class,
                                    StringDecoder.class,
                                    kafkaParams,
                                    topics);

  }
}
```

4.2 Hints

Once you have connected your Spark program to the Kafka broker you will have a DStream containing RDDs, each of which contains all the message strings in the batch length you supply during setup (see section 4.1). It will then be up to you to convert the JSON strings in each RDD into something that can be easily accesses in your chosen language. In Python you could use the inbuilt `JSON` library to convert each message string into a Python dictionary. In Java there are many JSON parsing libraries, you will have to modify your maven POM file to include the dependencies for the library of your choice.

To help in answering the queries in section 3, you should look at the section on Windowing in the Spark Streaming Programming Guide. If you set the window interval and slide interval to be equal you will get consecutive time sections of the DStream, which are required for this assignment. Sliding windows (where the slide interval is less than the window interval) have been known to cause issues with running Spark in local mode, as we are for this assignment.

4.3 Testing your programs

4.4 Setup your VM for streaming

There are a number of scripts and configuration files that need to be added to your VM in order for it to be able to stream events. SSH into your VM and run the following command:

```
$ wget https://s3-eu-west-1.amazonaws.com/csc8101-streaming-assignment-data/
    streaming-setup.sh
```

This should place the script `streaming-setup.sh` onto your VM. Then run the following command:

```
$ bash streaming-setup.sh
```

Once this script has finished running you should have a properly provisioned scripts folder and configured kafka server. You only need to run this script once and it can be deleted from your system once it has successfully executed.

4.4.1 Start the Kafka sever

When you first boot up you VM you will need to activate the Kafka message broker. There is a script called `kafka-setup.sh` in the `scripts` directory of your VM. Run the following command in your home directory (the one you are in when you first ssh into the VM) to start the broker:

```
$ bash scripts/kafka-setup.sh
```

This script will inform you when it is finished.

4.4.2 Streaming events to the broker

Once you have started the Kafka server you will need to begin sending data into Kafka to be read by your Spark program. To do this you will need to open a new terminal window on your own computer and ssh again into you VM. You should now have two windows open, both logged into your VM.

On one of the terminal windows run the following command when you are ready to stream events to the broker:

```
$ bash scripts/start-generator.sh
```

This script will read from the local events file and begin sending events to the broker. **Do not shut down this process (Ctrl-C) or close the terminal window** until your are finished running your Spark program. If you do, the world will not end, however events will stop streaming to the broker and your Spark program will produce no output. This is the reason you were asked to open another window into your VM in the step above, so that your could still interact with it.

4.4.3 Running your Spark program

In the other terminal window, that your opened above in section 4.4.2, you can now start your Spark streaming program. As with Assignment 1, this process varies depending on the programming language you have used:

Python If you have written a Python script containing you program, then this can be submitted directly to the `spark-submit` script on your VM. Use an scp command, like the one shown below, to copy the Python script file from your development machine to your VM.

```
$ scp -i <path-to-key-file>/csc8101-2015-2016-key.pem <path-to-Python-script>/
    distinct-licences.py ubuntu@<vm-dns-address>:<path-to-folder-on-VM>
```

Unlike Assignment 1 you need to include the Kafka connection library when you submit your script file to Spark. Below is an example of a call to the `spark-submit` script supplying the Kafka dependency for use in a Python based Spark program called `distinct-licences.py`:

```
$ bash ~/spark/bin/spark-submit --packages org.apache.spark:spark-streaming-
    kafka_2.10:1.6.0 distinct-licences.py
```

Java As with Assignment 1, an example maven POM file has been provided for you inside a sample project folder (the example project can be downloaded from here). Extract this to your development machine and create your Spark Streaming Java program in the `src/main/java` folder within this project folder. When you have completed your program, copy the project directory to you VM using scp. See below for an example command to do this:

```
$ scp -i <path-to-key-file>/csc8101-2015-2016-key.pem -rp <path-to-your-java-
    project> ubuntu@<vm-dns-address>:<path-to-folder-on-VM>
```

Once you have uploaded your project folder, you then need to ssh into your VM and navigate to the root of your Java project (this should be the folder where the POM file is). You then need to supply the following command to build a jar file:

```
$ mvn clean package
```

This will create a `<ProjectName>-0.0.1-jar-with-dependencies.jar` in the `target` folder within your project folder[1]. The jar with dependencies file can be then submitted directly to the `spark-submit` script. Below is an example of a call to the `spark-submit` script for the jar file created by the maven command above:

```
$ bash ~/spark/bin/spark-submit --class uk.ac.newcastle.CSC8101.sparkstreaming
    .DistinctLicences DistinctLicences.jar
```

4.4.4 Shutting down your VM

Once you have finished running your Spark program you can stop it in the terminal window in which it is running using the `Ctrl-C` keyboard command. You can stop the event generator script in the same way by switching to the other window in which it is running. Once both Spark and the generator script are no longer running you should change directory into the `scripts` folder and run the `kafka-stop.sh` script to shutdown the broker. You can then safely shutdown the VM.

[1] There will be another jar file within the `target` folder which does not contain the dependencies, you should ignore this file as it will not work correctly with Spark.