Lab2 - Spark Streaming and GraphX

1 Introduction

In this assignment you will learn how to (i) create a simple Spark Streaming application, while reading streaming data from Kafka, and (ii) work with GraphX to process graph data. In the rest of this document, we first explain how to install and test Kafka, and then explain the assignment.

2 Installing Kafka

We use Kafka as our data source, which is a distributed, partitioned, replicated commit log service. This section presents the steps you need to do to install Kafka on a Linux machine.

1. Download Kafka 2.0.0 from the following link
   https://archive.apache.org/dist/kafka/2.0.0/kafka_2.11-2.0.0.tgz

2. Set the following environment variables.

```
export KAFKA_HOME="/path/to/the/kafka/folder"
export PATH=$KAFKA_HOME/bin:$PATH
```

3. Kafka uses ZooKeeper to maintain the configuration information, so you need to first start a ZooKeeper server if you do not already have one.

```
$KAFKA_HOME/bin/zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties
```

4. Start the Kafka server.

```
$KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server.properties
```

5. Then, create a *topic*. A topic is a category or feed name to which messages are published. For each topic, the Kafka cluster maintains a partitioned log that looks like this. Let's create a topic named `avg` with a single partition and only one replica.

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1
  --topic avg
```

6. To see the list of topics you can run the following command.

```
$KAFKA_HOME/bin/kafka-topics.sh --list --zookeeper localhost:2181
```

7. Test Kafka by produce and consume some messages.

```
# Produce messages and send them to the topic "avg"
$KAFKA_HOME/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic avg

# Consume the messages sent to the topic "avg"
$KAFKA_HOME/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic avg --from-beginning
```

3 Spark-Kafka Integration

Spark Streaming can receive data from Kafka in two different approaches:

1. *Receiver-based* approach: in this approach, Spark receives data from Kafka through a receiver, and stores it in the Spark executors. Later, jobs launched by Spark Streaming can process the data. However, under default configuration, this approach can lose data under failures. To ensure zero-data loss, then, you have to additionally enable Write Ahead Logs (WAL) in Spark Streaming that synchronously saves all the received Kafka data into a distributed file system. To use this approach, you need to connect to Kafka through `KafkaUtils.createStream`.

```
val kafkaStream = KafkaUtils.createStream(streamingContext,
                  [ZK quorum], [consumer group id], [per-topic number of Kafka partitions to consume])
```

2. *Receiver-less* direct approach: in this approach, instead of using receivers to receive data, Spark periodically queries Kafka for the latest offsets in each topic+partition, and accordingly defines the offset ranges to process in each batch. Later, 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. In this approach, you can connect to Kafka by calling `KafkaUtils.createDirectStream`.

```
val kafkaStream = KafkaUtils.createDirectStream[
                  [key class], [value class], [key decoder class], [value decoder class]](
                  streamingContext, [map of Kafka parameters], [set of topics to consume])
```

In both models, you need to define the Kafka parameters, as a `Map` of configuration parameters to their values. The list of parameters are available here.

```
val kafkaConf = Map(
  "metadata.broker.list" -> "localhost:9092",
  "zookeeper.connect" -> "localhost:2181",
  "group.id" -> "kafka-spark-streaming",
  "zookeeper.connection.timeout.ms" -> "1000")
```

For Scala applications using sbt project definitions, you can link your streaming application with the following artifact.

```
libraryDependencies ++= Seq(
  "org.apache.spark" % "spark-streaming_2.11" % "2.4.3",
  "org.apache.spark" % "spark-streaming-kafka-0-8_2.11" % "2.4.3"
)
```

4 Task 1

In this assignment, you should implement a Spark Streaming application that reads streaming data from Kafka. The streaming data are (key, value) pairs in the form of `"String,int"`, and we want to calculate the average value of each key and continuously update it, while new pairs arrive. The results are in the form of (key, average value) pairs. To do this assignment, you can complete the given code in the `sparkstreaming` folder by first starting Kafka and creating a topic, named `avg` (as explained in Section 2).

```
zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties
kafka-server-start.sh $KAFKA_HOME/config/server.properties
kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic avg
```

Then, in the Spark Streaming code, use `mapWithState` to calculated the average value of each key in a statful manner. To test your code, run `sbt run` in the path of your code. To generate a streaming input (pairs of `"String,int"`) and feed them to Kafka, you are given a code in the `generator` folder. You just need to run the `sbt run` in its path. Here is an example output that `generator` produces:

```
value=z,19,
value=k,17,
value=x,23,
value=w,3,
value=n,14,
value=c,7,
value=g,15,
value=x,9,
value=q,10,
value=h,7,
value=h,10,
value=e,22,
value=t,5,
value=y,22,
value=q,2,
value=v,14,
```

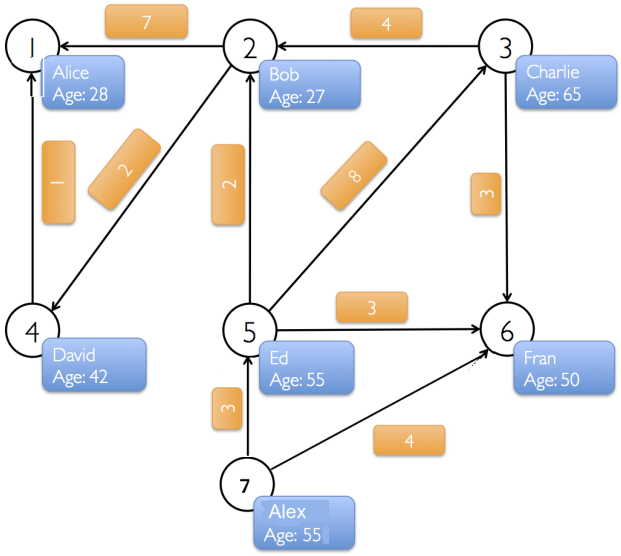
5 Task 2

In this assignment, you need to re-implement Task 1 but using Spark Structured Streaming. You should check, and modify, your `build.sbt` file to make sure you have the dependencies for Structured Streaming and the versions are correct based on your own configuration. The provided sbt files are just examples that will work for that specific task.

```
libraryDependencies ++= Seq(
  "org.apache.spark" % "spark-streaming_2.11" % "2.2.1",
  "org.apache.spark" % "spark-sql-kafka-0-10_2.11" % "2.2.1",
  "org.apache.spark" % "spark-streaming-kafka-0-8_2.11" % "2.2.1"
)
```

6 Task 3

In this task you will work with GraphX to process graph-based data. To do this assignment, write a code in GraphX to build the following graph and provide answer the questions. This graph shows a small social network with users and their ages modeled as vertices and likes modeled as directed edges.



1. Display the names of the users that are at least 30 years old.
2. Display who likes who.
3. If someone likes someone else more than 5 times than that relationship is getting pretty serious, so now display the lovers.
4. Print the number of people who like each user (e.g., Alice is liked by 2 people).

5. Print the names of the users who are liked by the same number of people they like (e.g., Bob and David).

6. Find the oldest follower of each user (hint: use the `aggregateMessages`).

What to deliver: you should submit the codes and write a short document to show your results. Please zip all your files in a single file with the filename format of `lab2_groupname.zip`.