Lab 2 - Stream and Graph Processing with Spark, Kafka, and Cassandra

Amir H. Payberah
payberah@kth.se

1 Introduction

In this lab assignment you will practice stream processing and graph processing using Apache Spark, Apache Kafka, and Apache Cassandra. The deliverable for this assignment should be a single zip file, as explained in the last section.

2 Part 1: Stream Processing

In this assignment you will learn how to create a simple Spark Streaming application, while reading streaming data from Kafka and storing the result in Cassandra datastore. Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming provides a high-level abstraction called *discretized stream* or `DStream` that represents a continuous stream of data. `DStream` can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other `DStream`. Internally, a `DStream` is represented as a sequence of `RDD`s. In the following sections, we first explain how to install and test Kafka and Cassandra, and then explain the assignment for this part.

3 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 go through in order to install Kafka on a Linux machine.

1. Download Kafka (binary) from the following link, and extract it.
   https://kafka.apache.org/downloads

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. Keep in mind that you might have to change the command arguments based on your custom configuration.

```
$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 producing and consuming 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
```

4 Spark-Kafka Integration

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

1. *Receiver-based* approach: in this approach, Spark receives data from Kafka through a receiver, and stores it in 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"
)
```

5 Installing Cassandra

Here, we show how to install and test Cassandra.

1. Download Cassandra from the following link
   https://archive.apache.org/dist/cassandra/3.11.2/apache-cassandra-3.11.2-bin.tar.gz
2. Download and install Anaconda. For using `cqlsh`, you will need **Python 2.7**. You can download it from the following link:
   https://repo.anaconda.com/archive/Anaconda2-5.2.0-Linux-x86_64.sh
3. Set the following environment variables.

```
export CASSANDRA_HOME="/path/to/the/cassandra/folder"
export PYTHONPATH="/path/to/the/python/folder"
export PATH=$PYTHONPATH/bin:$CASSANDRA_HOME/bin:$PATH
```

4. Start Cassandra in the foreground.

```
$CASSANDRA_HOME/bin/cassandra -f
```

5. Start the `cqlsh` prompt.

```
$CASSANDRA_HOME/bin/cqlsh
```

6. Now, let's create a *keyspace*. A keyspace is similar to a schema/database in the RDBMS world. To create a keyspace execute the following CQL command. The `WITH REPLICATION` part of the command states that the `wordcount_keyspace` keyspace should use a simple replication strategy and will only have one replica for all data inserted into the keyspace.

```
create keyspace wordcount_keyspace
      with replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

7. Print the list of keyspaces in Cassandra.

```
describe keyspaces;
```

8. Now let's create a column family. In order to create a column family, first you will need to navigate to the `wordcount_keyspace` keyspace. Then, create the `Words` table.

```
use wordcount_keyspace;
create table Words (word text, count int, primary key (word));
```

9. Insert a row into the above column family.

```
insert into Words(word, count) values('hello', 5);
```

10. Now, let's look at the table now.

```
select * from Words;
```

Let's examine what happened as a result of creating and inserting a row into the `Words` table. This will require us to flush data from a memtable to disk thus creating an SSTable on disk. We will use a utility called `nodetool` to help us flush data to disk.

```
$CASSANDRA_HOME/bin/nodetool flush wordcount_keyspace
```

Here is how Cassandra stores data (all keyspace and SSTable related data) on disk. By default it stores data at `data/data` in `$CASSANDRA_HOME`. However you can change it, by updating the `data_file_directories` at `$CASSANDRA_HOME/conf/cassandra.yaml`. The directory structure and component files have the following structure:

- `Data.db`: this is the base data file for the SSTable. All other SSTable related files can be generated from this file.
- `CompressionInfo.db`: it holds information about the uncompressed data length.
- `Filter.db`: the serialized bloom filter.
- `Index.db`: an index to the row keys with pointers to their position in the data file.
- `Summary.db`: SSTable index summary.
- `Statistics.db`: statistical metadata about the content of the SSTable.
- `TOC.txt`: a file which contains a list of files outputted for each SSTable.

Now, we can see what the underlying format looks like. The `sstabledump` is a utility that can be used to convert a binary SSTable file into a JSON. Let's convert data inserted into our `Words` table into JSON. You should replace the `<NUMS>` with what you see on your machine.

```
chmod +x $CASSANDRA_HOME/tools/bin/sstabledump
$CASSANDRA_HOME/tools/bin/sstabledump $CASSANDRA_HOME/data/data/wordcount_keyspace/words-<NUMS>/mc-1-big-Data.db
```

6 Spark-Cassandra Integration

A Spark application can connect to Cassandra using the following commands.

```
import org.apache.spark.sql.cassandra._
import com.datastax.spark.connector._
import com.datastax.driver.core.{Session, Cluster, Host, Metadata}
import com.datastax.spark.connector.streaming._

val cluster = Cluster.builder().addContactPoint("127.0.0.1").build()
val session = cluster.connect()
```

To execute a command on a connected Cassandra instance, you can use the `execute` command as below.

```
session.execute("CREATE KEYSPACE ...")
session.execute("CREATE TABLE ...")
```

And, to store data from RDD to Cassandra you can use the `saveToCassandra` command.

```
rdd.saveToCassandra("wordcount_keyspace", "words", SomeColumns("word", "count"))
```

At the end and after finishing the work, you should close the connection.

```
session.close()
```

For Scala applications using SBT project definitions, add the following dependencies in `build.sbt`.

```
libraryDependencies ++= Seq(
  ("com.datastax.spark" %% "spark-cassandra-connector" % "2.4.0").exclude("io.netty", "netty-handler"),
  ("com.datastax.cassandra" % "cassandra-driver-core" % "3.0.0").exclude("io.netty", "netty-handler")
)
```

7 Your Assignment for Part 1

In the first part of this lab assignment, you should implement a Spark Streaming application that reads streaming data from Kafka, and stores the results in Cassandra. 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. We would also like to store the result in Cassandra continuously. The results are in the form of (key, average value) pairs.

To do this part, first you need to start Kafka and create a topic, named `avg` (as explained in Section 3).

```
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
```

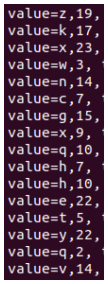
You also need to start Cassandra

```
cassandra -f
```

Then, in the Spark Streaming code, first build a keyspace and a table in Cassandra, using `session.execute` command. Call the keyspace `avg_space` and the table `avg`, in which the table has two columns of types `text` and `float`, for the key and the average value, respectively.

```
session.execute("CREATE KEYSPACE IF NOT EXISTS avg_space WITH REPLICATION =
                { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };")
session.execute("CREATE TABLE IF NOT EXISTS avg_space.avg (word text PRIMARY KEY, count float);")
```

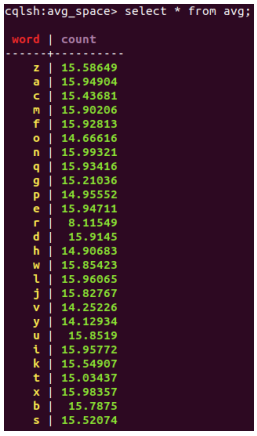
In the rest of your code, use `mapWithState` to calculated the average value of each key in a statful manner, and call `saveToCassandra` to store the result in Cassandra. To test your code, run `sbt run` in the path of your code. To generate a streaming input (pairs of `"String,int"`) and feed them to 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:



To test if the results are correctly stored in Cassandra, start the `cqlsh` in a terminal, and go to the `avg_space` keyspace, and print the content of the `avg` table.

```
use avg_space;
select * from avg;
```

You should see some results as below:



8 Part 2: Graph Processing

In this lab you will work with GraphX to process graph-based data. The programming environment is the same as the one you used in the previous lab assignment. To do this assignment, extract the given zip file and copy the Notebooks, `figs`, and the `data` folder from `src/notebook` to the folder you have started the Jupyter Notebook. Then, you should be able to see the two files `graphx_social_network.ipynb` and `graphx_songs.ipynb` on your browser on the address `localhost:8888`. The files are self-explanatory that describe what you need to do.

9 Your Assignment for Part 2

Follow the instructions in the notebooks and complete `graphx_social_network.ipynb` and `graphx_songs.ipynb`.

10 What to Deliver

You should complete the `KafkaSpark.scala` code, and write a short document to explain how you implemented your code and how to run it, and also show your results (with screenshots). Also, for the second part of the assignment, you should complete the given Jupyter notebooks. Please zip all your files in **a single file** with the filename format of `lab2_groupname.zip` and upload it on Canvas.