Lab2 - ID2221 HT20-1 Data-Intensive Computing

-Mattia Zoffoli & Varrun Varatharajan

Part 1 – SparkStreaming

How to execute the code

First we need to start a zookeeper server:

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

Then we need to start the kafka server in a new terminal:

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

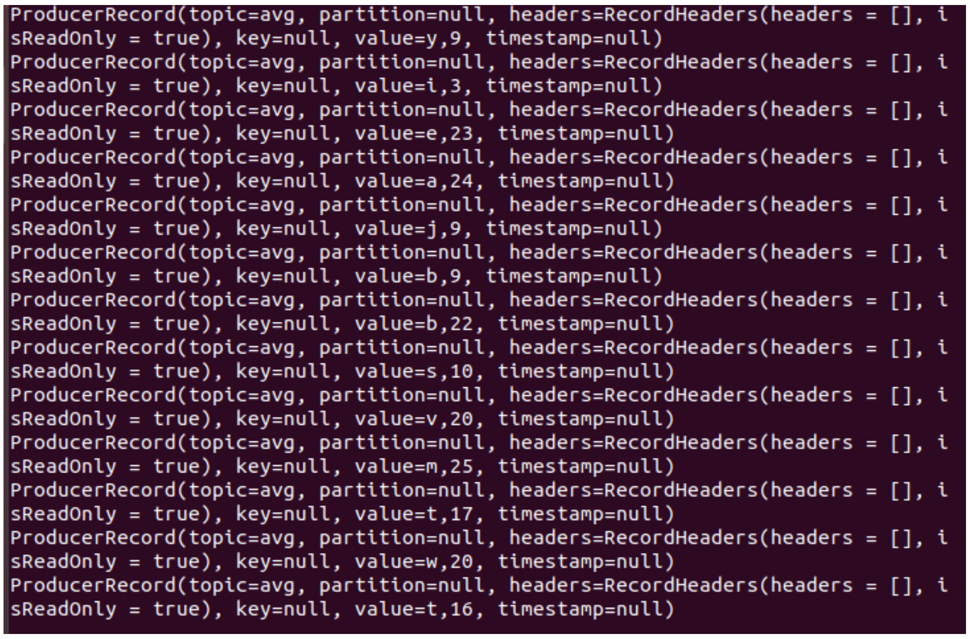
Then we need to create an "avg" topic in a new terminal, and we can run Cassandra there in the foreground:

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

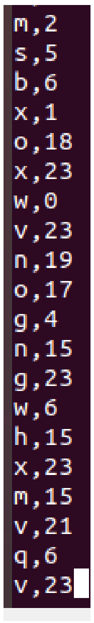
Then we should navigate to the generator folder and run the generator with sbt:

```
sbt run
```

In this way we see the stream of data generated looking like the images below on the left.



```
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=y,9, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=i,3, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=e,23, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=a,24, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=j,9, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=b,9, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=b,22, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=s,10, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=v,20, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=m,25, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=t,17, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=w,20, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(headers = [], i
sReadOnly = true), key=null, value=t,16, timestamp=null)
```



```
m,2
s,5
b,6
x,1
o,18
x,23
w,0
v,23
n,19
o,17
g,4
n,15
g,23
w,6
h,15
x,23
m,15
v,21
q,6
v,23
```

The second image is generated by consuming the stream of data running the following command in a new terminal:

```
$KAFKA_HOME/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --
topic avg --from-beginning
```

Finally we can navigate to the sparkstreaming folder and run our code with sbt:

```
sbt run
```

The final result will be stored in a Cassandra table, which can be consulted by opening a clqsh prompt with:

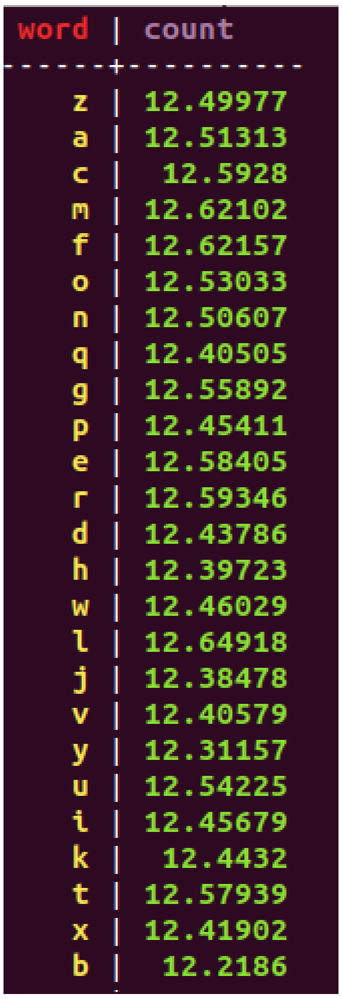
```
$CASSANDRA_HOME/bin/cqlsh
```

then, from inside the prompt, we can activate the avg_space keyspace, and visualizing the avg topic:

```
use wordcount_keyspace;

select * from Words;
```

The final result should look similar to the image on the right.



```
word | count
------+-----------
    z | 12.49977
    a | 12.51313
    c |  12.5928
    m | 12.62102
    f | 12.62157
    o | 12.53033
    n | 12.50607
    q | 12.40505
    g | 12.55892
    p | 12.45411
    e | 12.58405
    r | 12.59346
    d | 12.43786
    h | 12.39723
    w | 12.46029
    l | 12.64918
    j | 12.38478
    v | 12.40579
    y | 12.31157
    u | 12.54225
    i | 12.45679
    k |  12.4432
    t | 12.57939
    x | 12.41902
    b |  12.2186
```

Code summary

Initially we import all the libraries necessary to use Cassandra, stream processing and kafka.

```
package sparkstreaming

import java.util.HashMap
import org.apache.kafka.clients.consumer.ConsumerConfig
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka._
import kafka.serializer.{DefaultDecoder, StringDecoder}
import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._
import org.apache.spark.storage.StorageLevel
import java.util.{Date, Properties}
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerRecord,
ProducerConfig}
import scala.util.Random

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._
```

Inside our main, we define a spark cluster, session, and a keyspace avg_space in Cassandra with a topic avg inside this keyspace.

```
object KafkaSpark {
  def main(args: Array[String]) {
    // connect to Cassandra and make a keyspace and table as explained in the
document
    val cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
    val session = cluster.connect()
    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);")
```

Now, still inside main, we create a Direct Stream of data, reading from Kafka Stream. The spark straming context is set to check for updates ever 2 seconds.
Once we have a stream of raw messages we can extract the information we need from it as a pair of keys and values [String, Double].

```
    // make a connection to Kafka and read (key, value) pairs from it
    // set local[2] because n of streaming sources is 1
    val conf = new SparkConf().setMaster("local[2]").setAppName("KafkaStream")

    // define spark streaming context
    val ssc = new StreamingContext(conf, Seconds(2))
    ssc.checkpoint("checkpoint")

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

    val messages = KafkaUtils.createDirectStream[String, String, StringDecoder,
StringDecoder](ssc, kafkaConf, Set("avg"))

    // get record
    val value = messages.map{case (key, value) => value.split(',')}
    // get pairs
    val pairs = value.map{record => (record(1), record(2).toDouble)}
```

We can now process the stream of data to compute the average value of each key, memorizing the previous state and updating them when new data comes from the stream. The average value is computed for each key as the sum of previous and current value, divided by the total number of occurrencies of that key.

```
    // measure the average value for each key in a stateful manner
    def mappingFunc(key: String, value: Option[Double], state:
State[Tuple2[Double, Int]]): (String, Double) = {
        val oldState:Tuple2[Double, Int] = state.getOption.getOrElse((0.0, 0))
        val sum:Double = oldState._1
        val count:Int = oldState._2
        val newSum = value.getOrElse(0.0) + sum
        val newCount = count + 1
        state.update((newSum, newCount))
        (key, newSum/newCount)
    }

    val stateDstream = pairs.mapWithState(StateSpec.function(mappingFunc _))
```

Finally, we store the result in the cassandra table, using the keyspace avg_space, in the topic avg, we start the sparkstreaming context, which was not initialized until now, but just defined, and we wait for a termination signal to stop the loop.

```
    // store the result in Cassandra
    stateDstream.saveToCassandra("avg_space", "avg", SomeColumns("word",
"count"))

    ssc.start()
    ssc.awaitTermination()
  }
}
```