Group 50
Leonardo Remondini 19970128-T512
Haris Poljo 19970731-6213

Lab 2 - Part 1

In this first part of the assignment, we implemented 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.

How to run the code

Unzip the zip file and (inside the unzipped folder) go through the following steps to compile and run the code

1. Set the following environment variables.

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

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

2. Start Kafka and create a topic, named `avg` (If it does not exist).

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

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

3. Start Cassandra too.

```
cassandra -f
```

4. To test the code, run `sbt run` in the path of the code.

5. To generate a streaming input and it to Kafka, run `sbt run` in the generator folder

6. Look at the results stored in Cassandra

```
$CASSANDRA_HOME/bin/cqlsh
use avg_space;
select * from avg;
```

Cassandra Configuration

A Cassandra connection is created as follows

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

then, we build a keyspace and a table in Cassandra, using `session.execute` command.

At the bottom of the code, stream data is continiously updated and stored in Cassandra using `saveToCassandra` function.

Spark Streaming configuration

Firstly a spark streaming context is created, which is also checkpointed because our map function will be stateful.

Then a stream from Kafka is created by using `KafkaUtils.createDirectStream`, which takes 3 arguments, firstly the spark streaming context, configurations to connect to Kafka which has the following form:

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

and lastly, we define what topic should be fetched which in this case is “avg”.

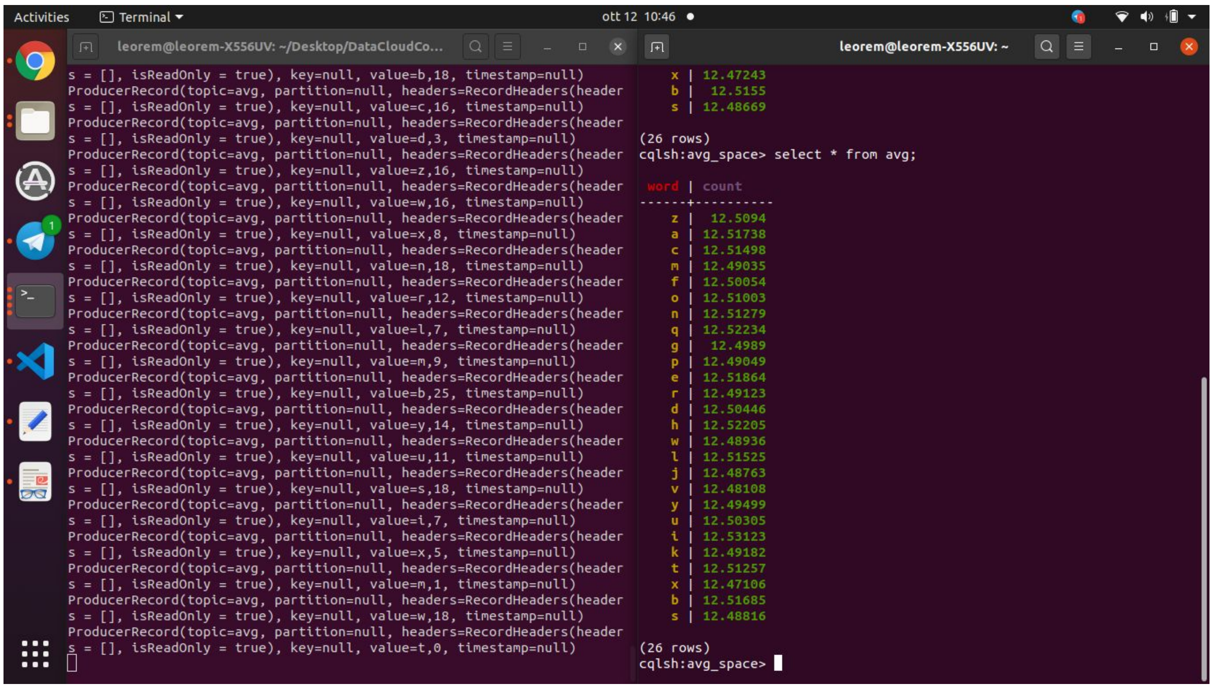
Preprocess

The values that are received from the stream aren't in the right form to be passed to the map function. The key and value are stored as a string with a whitespace separating them. Therefore a tuple is created by splitting the string on whitespace and then creating the tuple. The value needs to be cast to double to be able to be used in the map function.

Mapping function

The function gets the current key and value as input, and we also have a state which is a tuple of the form (Double, Int), the state will store the current total sum of the stream and the count of the total values processed. When the map function is called it will first fetch the state. Then it will calculate the new sum given the new value and then calculate the new count. The state will then be updated. Lastly, the map function will return a tuple, with the key and the current average value. The average value is calculated by dividing the total sum with the count.

Results (screenshot of results stored in Cassandra)



```
s = [], isReadOnly = true), key=null, value=b,18, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=c,16, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=d,3, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=z,16, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=w,16, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=x,8, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=n,18, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=r,12, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=l,7, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=m,9, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=b,25, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=y,14, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=u,11, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=s,18, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=i,7, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=x,5, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=m,1, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=w,18, timestamp=null)
ProducerRecord(topic=avg, partition=null, headers=RecordHeaders(header
s = [], isReadOnly = true), key=null, value=t,0, timestamp=null)
```

```
    x | 12.47243
    b |  12.5155
    s | 12.48669

(26 rows)
cqlsh:avg_space> select * from avg;

 word | count
------+----------
    z |  12.5094
    a | 12.51738
    c | 12.51498
    m | 12.49035
    f | 12.50054
    o | 12.51003
    n | 12.51279
    q | 12.52234
    g |  12.4989
    p | 12.49049
    e | 12.51864
    r | 12.49123
    d | 12.50446
    h | 12.52205
    w | 12.48936
    l | 12.51525
    j | 12.48763
    v | 12.48108
    y | 12.49499
    u | 12.50305
    i | 12.53123
    k | 12.49182
    t | 12.51257
    x | 12.47106
    b | 12.51685
    s | 12.48816

(26 rows)
cqlsh:avg_space>
```