Data Intensive Computing - Lab 2 Part 1

Sri & Nik - 6th Oct, 2019

Code Implementation

Initialization - Cassandra, Kafka, Spark

In the main function found in the *KafkaSpark* object*,* we create a *cluster* and connect our *session* to Cassandra. In this *session,* we create a table called *avg* within the keyspace *avg_space.* We define the *KafkaConf* to *Map* the topics*.* We also create a Spark *StreamingContext* defining its configuration, number of threads to allocate, and its application name.

DStream

Using the *KafkaConf* and *StreamingContext,* we create Kafka *DStream* that consumes topic *avg,* with the *key*, *value* and the corresponding *decoder* classes as *Strings*. The *messages* received through this stream are split at ‘,’. And the *value* in the message is further split at ‘.’. As the first value is the key (letter) which is a *String* it remains as it is and the second value that is the count is cast to *Double* and stored as *pairs.*

Mapping Function

With the above done, we have the data in the format and structure we need it in. Now we define *mappingFunc* which is used to *map* the streamed data and compute the average count on the fly. We change the state to hold two variables one for the *average* and one for the length of values used to calculate the average as we cannot calculate the new average without knowing the previous count.

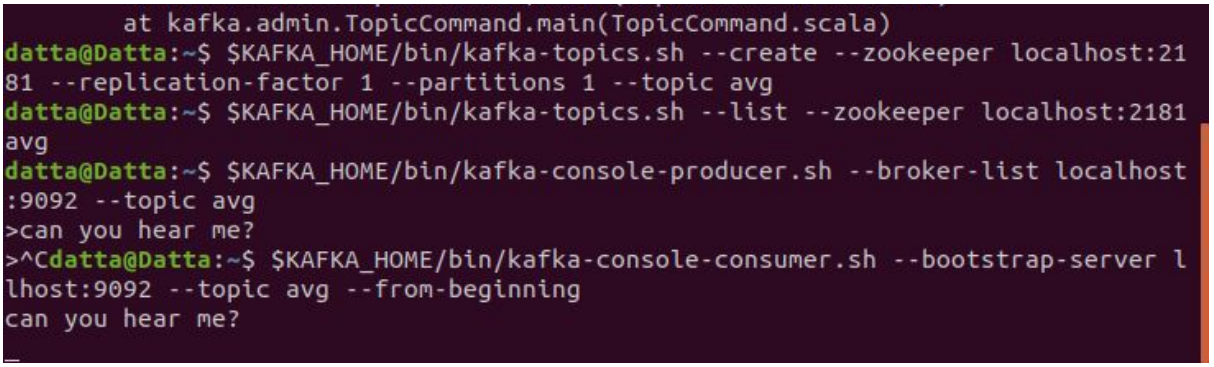
If the state exists, not timing out and value is defined, that is if everything is OK, we calculate the new average, *average_updated* and new count *count_updated* as we cannot reassign values to the variables. We update the *state* with a new state, *state_updated* and return the new state. If the value is defined but a state does not exist, then it is the first data point so we initialize a state with count as 1. If value is also not defined then there is some error.

MapWithState

StateDStream contains the result of the *pairs,* mapped with *mapWithState* using *mappingFunc.* The results are saved to Cassandra in avg_space and avg table in their respective columns. We also create a checkpoint for the session to recover from in case of failure.

Code Execution

We have to parallelly start Cassandra, ZooKeeper, KakfaServer and create *avg* topic and can check the communication as in the screenshot below.

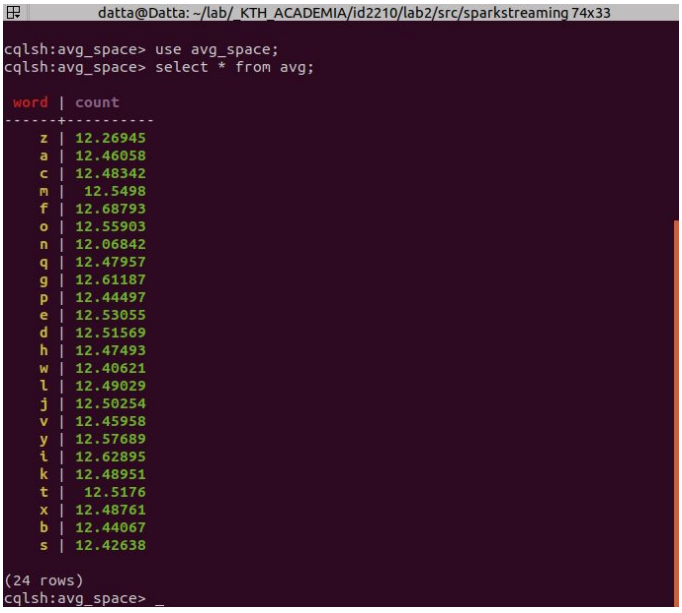


Next, to get the input stream, run the generator using `sbt run` where *Producer.scala* is located. The output should be as in the screenshot below.

Run `sbt run` where *KafkaSpark.scala* is located to compute the average. The output can be dispayed in Cassandra using

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

The output is as in the screenshot below.



Other things mentioned in the instructions

Check Cassandra keyspaces and create tables and insert data.

```
cqlsh> use wordcount_keyspace;
cqlsh:wordcount_keyspace> create table Words (word text, count int, primar
y key (word));
cqlsh:wordcount_keyspace> insert into Words(word, count) values('hello', 5
);
cqlsh:wordcount_keyspace> select * from Words;

 word  | count
-------+-------
 hello |     5

(1 rows)
cqlsh:wordcount_keyspace>
```

Convert a binary SSTable file into a JSON

```
datta@Datta:~$ $CASSANDRA_HOME/tools/bin/sstabledump $CASSANDRA_HOME/data/
data/wordcount_keyspace/words-f8dd6d20e77d11e98b01539ff60a9dc3/mc-1-big-Da
ta.db
[
  {
    "partition" : {
      "key" : [ "hello" ],
      "position" : 0
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 28,
        "liveness_info" : { "tstamp" : "2019-10-05T14:40:13.071076Z" },
        "cells" : [
          { "name" : "count", "value" : 5 }
        ]
      }
    ]
  }
]datta@Datta:~$
```