KUNGL TEKNISKA HÖGSKOLAN

MSc Machine Learning

ID2221 Data Intensive Computing

Lab 3: Spark Streaming, Kafka and Casssandra

Author:

Alexandros Ferles ferles@kth.se

George Zervakis zervakis@kth.se

Professor:

Amir H. Payberah

payberah@kth.se

Contents

1 Top Ten Users

1.1 Problem Description

In this assignment, we were instructed to implement a Spark Streaming application, that should:

1. Read key-value pairs as streaming data from Kafka. Keys in this setting may be any letter of the English alphabet, and the corresponding value of each key is a positive integer.
2. Compute the average of values paired to each letter in an online manner.
3. Store and update periodically the these average values in Cassandra datastore.

1.2 Technical decisions

During this setting, some technical decisions were to be made:

- **Choose the data receiving approach**. When designing the streaming procedure between Kafka and Spark Streaming, the designer can choose beteween the *Receiver-based* approach and the *Receiver-less* direct approach. A lot of analysis is made in comparing these two approaches[1]. To summarize some of this analysis, the former needs extra tweeking to avoid loss of data, while the latter ensures data-loss and less code-optimization from the side of the designer of the application. Thus, we chose to proceed with the **Receiver-less** based approach.
- **Re-designing the signature of the mappingFunc**. The state of the mapping function *mappingFunc* was originally in the form State[Double], which meant that a track of the average value was to be kept and updated periodically. However, in order to efficiently compute the average value of its key (letter), we needed both the new values and a track of the occurrences of each key, so that we can sum all the values and divide by this number (occurrences). Hence, we changed the state type to *state: State[(Int, Int)]*, so that we can keep track of both the sum of the values that correspond to each key and the count of the distinct values that have occurred up to a point.

1.3 Streaming Application

The behavior of the application can be sumarized to the following points:

- **Create a topic in Kafka**. This way, we have defined the destination where messages can be sent and read from.
- **Connect to Cassandra from Spark application**. A session has to be made between the Streaming application and the Cassandra datastore, so that the keyspace and tables can be created and updated periodically with the results.
- **Receive messages through DirectStream**. The implementation decision of this step are already described in (1.2)

[1] `https://spark.apache.org/docs/1.6.1/streaming-kafka-integration.html`

- **Process the received messages into an appropriate structure**. The application receives key-value pairs where the key is empty, and the values is a string unification of a letter of the alphabet and its value (e.g ”z,12”). We need to separate these values and form a new key-value pair in the form of {*key*: alphabet letter, *value*: integer value} (e.g {z, 12}). The function *help_split* helps us do that, by splitting the received string using the ”,” character as separator, and creating a tuple using the letter and the the integer form of the value as first and second element of this tuple respectively.
- **State update and computation of average value**. As already stated in (1.2), the state keeps track of two integer values: i) The sum of all the received values and ii) a count of how many distinct values have already received for each letter. This way, the mapping function can easily compute the average value by performing a simple division using the sum as nominator and the count as denominator. A key-value pair in the form(letter, average value) is returned.

  Save the updates to Cassandra. All updated key-value pairs are stored in the Cassandra datastore.

1.4 Execution

In order to be able to test this implementation, the reader is expected to have already Spark, Kafka and Cassandra installed and also have configured them with local settings. If this is not the case, you can find instructions in [1], listed as 'lab_description.pdf' in this very report structure.

The reader should also ensure that the zookeper and kafka servers are running, the Kafka topic ”avg” is already created, and lastly, that Cassandra is already started. In order to do so, please execute the following commands in separate terminals:

```
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
cassandra -f
```

After everything is in place, you should start the application. To generate an endless stream of messages, navigate to */src/generator* and execute *sbt run.* Then navigate to *src/sparkstreaming/* and also execute *sbt run*[2].

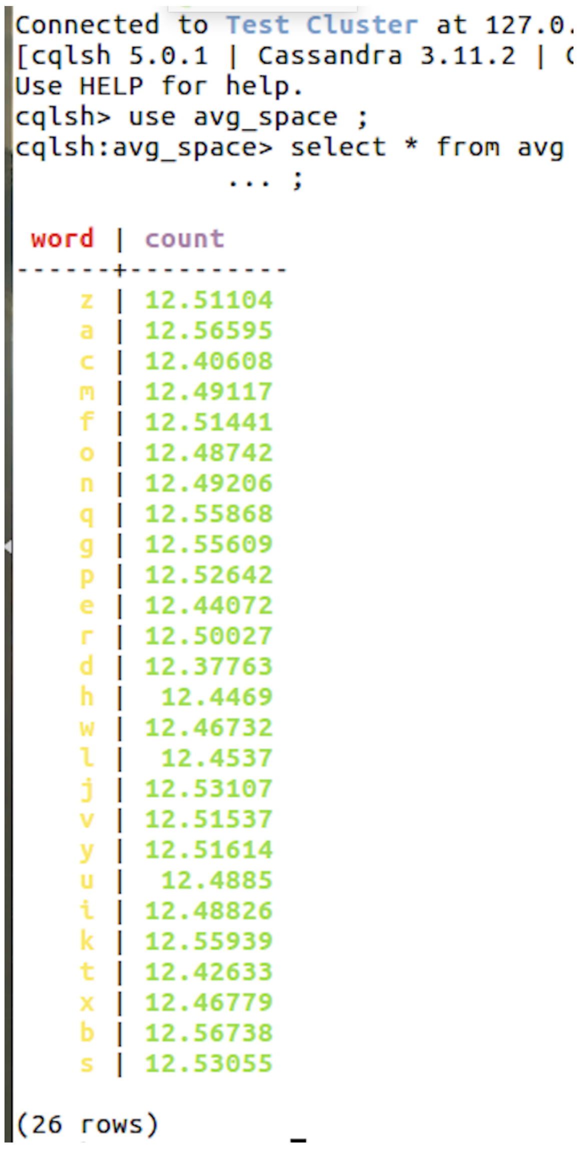
After a few seconds, results will be stored in Cassandra. Simply execute *cqlsh* and print the contents of the *avg* table of *avg_space* keyspace:

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

[2]You can also just test that everything is ok with the source code by executing i*sbt compile* in both paths

1.5 Results

A typical view of the *avg* table should be similar to the following:



```
Connected to Test Cluster at 127.0.
[cqlsh 5.0.1 | Cassandra 3.11.2 | (
Use HELP for help.
cqlsh> use avg_space ;
cqlsh:avg_space> select * from avg
             ... ;

 word | count
------+----------
    z | 12.51104
    a | 12.56595
    c | 12.40608
    m | 12.49117
    f | 12.51441
    o | 12.48742
    n | 12.49206
    q | 12.55868
    g | 12.55609
    p | 12.52642
    e | 12.44072
    r | 12.50027
    d | 12.37763
    h |  12.4469
    w | 12.46732
    l |  12.4537
    j | 12.53107
    v | 12.51537
    y | 12.51614
    u |  12.4885
    i | 12.48826
    k | 12.55939
    t | 12.42633
    x | 12.46779
    b | 12.56738
    s | 12.53055

(26 rows)
```

Figure 1: Displaying the stored results in Cassandra

Screenshots of more results can be found in the *results* folder.

References

[1] A. H. Payberah. Lab3 - spark streaming, kafka and cassandra. 2018.