Stream Processing Exericse 3 - Creating an Streaming Job

January 15, 2021

0.1 Stream Processing Exercise 3 - Creating an Streaming job

Purpose of this exercise is to create your first Streaming job using Spark Streaming and reading from a Network socket.

First logon inside jupyter container with following command:

Launch netcat server

Let's code our Spark streaming job. First we import the names of the Spark Streaming classes and some implicit conversions from StreamingContext into our environment in order to add useful methods to other classes we need (like DStream). StreamingContext is the main entry point for all streaming functionality. We create a local StreamingContext with two execution threads, and a batch interval of 5 seconds.

[1]:
```
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(appName="PythonStreamingNetworkWordCount")

ssc = StreamingContext(sc, 5)
```

Using this context, we can create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

[2]:
```
lines = ssc.socketTextStream("localhost", 9999)
```

This lines DStream represents the stream of data that will be received from the data server. Each record in this DStream is a line of text. Next, we want to split the lines by space characters into words.

flatMap is a one-to-many DStream operation that creates a new DStream by generating multiple new records from each record in the source DStream. In this case, each line will be split into multiple words and the stream of words is represented as the words DStream. Next, we want to count these words.

[3]:
```
counts = lines.flatMap(lambda line: line.split(" "))\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)
```

```
counts.pprint()
```

The words DStream is further mapped (one-to-one transformation) to a DStream of (word, 1) pairs, which is then reduced to get the frequency of words in each batch of data. Finally, wordCounts.print() will print a few of the counts generated every second.

Note that when these lines are executed, Spark Streaming only sets up the computation it will perform when it is started, and no real processing has started yet. To start the processing after all the transformations have been setup, we finally call

[4]:
```
ssc.start()
ssc.awaitTermination()
```

```
-------------------------------------------
Time: 2021-01-15 18:49:40
-------------------------------------------
('hola', 1)

-------------------------------------------
Time: 2021-01-15 18:49:45
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:49:50
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:49:55
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:50:00
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:50:05
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:50:10
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:50:15
-------------------------------------------
('hola', 1)
('Vicent', 1)
```

```
-------------------------------------------
Time: 2021-01-15 18:50:20
-------------------------------------------
('guay', 1)
('mola', 1)

-------------------------------------------
Time: 2021-01-15 18:50:25
-------------------------------------------
('streaming', 1)
('en', 1)
('Spark', 1)

-------------------------------------------
Time: 2021-01-15 18:50:30
-------------------------------------------

-------------------------------------------
Time: 2021-01-15 18:50:35
-------------------------------------------
('streaming', 1)
('en', 1)
('Spark', 1)
('Vicent', 1)
(' es', 1)
('per', 1)
('una', 1)
('locura', 1)

-------------------------------------------
Time: 2021-01-15 18:50:40
-------------------------------------------



      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-4-18f3db416f1c> in <module>
         1 ssc.start()
   ----> 2 ssc.awaitTermination()


       /usr/local/spark/python/pyspark/streaming/context.py in␣
↪awaitTermination(self, timeout)
```

```
      190         """
      191         if timeout is None:
  --> 192             self._jssc.awaitTermination()
      193         else:
      194             self._jssc.awaitTerminationOrTimeout(int(timeout * 1000))


      /usr/local/spark/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in␣
↪__call__(self, *args)
     1253             proto.END_COMMAND_PART
     1254
  -> 1255         answer = self.gateway_client.send_command(command)
     1256         return_value = get_return_value(
     1257             answer, self.gateway_client, self.target_id, self.name)


      /usr/local/spark/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in␣
↪send_command(self, command, retry, binary)
      983         connection = self._get_connection()
      984         try:
  --> 985             response = connection.send_command(command)
      986             if binary:
      987                 return response, self.
↪_create_connection_guard(connection)


      /usr/local/spark/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in␣
↪send_command(self, command)
     1150
     1151         try:
  -> 1152             answer = smart_decode(self.stream.readline()[:-1])
     1153             logger.debug("Answer received: {0}".format(answer))
     1154             if answer.startswith(proto.RETURN_MESSAGE):


      /opt/conda/lib/python3.7/socket.py in readinto(self, b)
      587         while True:
      588             try:
  --> 589                 return self._sock.recv_into(b)
      590             except timeout:
      591                 self._timeout_occurred = True


      KeyboardInterrupt:
```

[ ]: