Spark Streaming

This little demo uses Spark DStreams to ingest words from a stream, determine how long the words are, and then plots the distribution of string lengths over time. You can try feeding in different books from, say, Project Gutenberg to see the distribution change with books from different periods.

Note: see `streamer.sh` for an example of a small program that streams out individual words.

In [23]:
```
from pyspark.streaming import StreamingContext
# The "1" here is the number of seconds between microbatches:
ssc = StreamingContext(sc, 1)

# Required to be able to do state updates:
ssc.checkpoint("checkpoint")
```

In [24]:
```
# Assumes the stream is running on the same machine as the driver.
# That's not very common, so you'll probably change 'localhost'
# to something else. In fact, using 'localhost' even from the local
# machine seems to be hit or miss.
sock = ssc.socketTextStream("orion03", 9999)
```

In [28]:
```
# Running this will start listening:
ssc.start()
```

```
23/05/20 23:12:03 WARN TaskSchedulerImpl: Initial job has not accepted any res
ources; check your cluster UI to ensure that workers are registered and have s
ufficient resources
23/05/20 23:12:18 WARN TaskSchedulerImpl: Initial job has not accepted any res
ources; check your cluster UI to ensure that workers are registered and have s
ufficient resources
```

```
ERROR:root:KeyboardInterrupt while sending command.
Traceback (most recent call last):
  File "/bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4
j/java_gateway.py", line 1038, in send_command
    response = connection.send_command(command)
  File "/bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4
j/clientserver.py", line 511, in send_command
    answer = smart_decode(self.stream.readline()[:-1])
  File "/home2/anaconda3/lib/python3.10/socket.py", line 705, in readinto
    return self._sock.recv_into(b)
KeyboardInterrupt
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[28], line 2
      1 # Running this will start listening:
----> 2 ssc.start()

File /bigdata/spark-3.3.2-bin-hadoop3/python/pyspark/streaming/context.py:214,
in StreamingContext.start(self)
    210 def start(self) -> None:
    211     """
    212     Start the execution of the streams.
    213     """
--> 214     self._jssc.start()
    215     StreamingContext._activeContext = self

File /bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4j/ja
va_gateway.py:1320, in JavaMember.__call__(self, *args)
   1313 args_command, temp_args = self._build_args(*args)
   1315 command = proto.CALL_COMMAND_NAME +\
   1316     self.command_header +\
   1317     args_command +\
   1318     proto.END_COMMAND_PART
-> 1320 answer = self.gateway_client.send_command(command)
   1321 return_value = get_return_value(
   1322     answer, self.gateway_client, self.target_id, self.name)
   1324 for temp_arg in temp_args:

File /bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4j/ja
va_gateway.py:1038, in GatewayClient.send_command(self, command, retry, binar
y)
   1036 connection = self._get_connection()
   1037 try:
-> 1038     response = connection.send_command(command)
   1039     if binary:
   1040         return response, self._create_connection_guard(connection)

File /bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4j/cl
ientserver.py:511, in ClientServerConnection.send_command(self, command)
    509 try:
    510     while True:
--> 511         answer = smart_decode(self.stream.readline()[:-1])
    512         logger.debug("Answer received: {0}".format(answer))
    513         # Happens when a the other end is dead. There might be an empt
y
    514         # answer before the socket raises an error.

File /home2/anaconda3/lib/python3.10/socket.py:705, in SocketIO.readinto(self,
b)
    703 while True:
    704     try:
--> 705         return self._sock.recv_into(b)
    706     except timeout:
    707         self._timeout_occurred = True

KeyboardInterrupt:
```

In [20]:
```
# IMPORTANT: you need the stopSparkContext=False, otherwise
# your driver will die and you'll have to restart Jupyter
ssc.stop(stopSparkContext=False)
```

```
23/05/20 15:36:53 WARN TaskSchedulerImpl: Initial job has not accepted any res
ources; check your cluster UI to ensure that workers are registered and have s
ufficient resources
23/05/20 15:37:08 WARN TaskSchedulerImpl: Initial job has not accepted any res
ources; check your cluster UI to ensure that workers are registered and have s
ufficient resources
```

```
ERROR:root:KeyboardInterrupt while sending command.
Traceback (most recent call last):
  File "/bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4
j/java_gateway.py", line 1038, in send_command
    response = connection.send_command(command)
  File "/bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4
j/clientserver.py", line 511, in send_command
    answer = smart_decode(self.stream.readline()[:-1])
  File "/home2/anaconda3/lib/python3.10/socket.py", line 705, in readinto
    return self._sock.recv_into(b)
KeyboardInterrupt
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[20], line 3
      1 # IMPORTANT: you need the stopSparkContext=False, otherwise
      2 # your driver will die and you'll have to restart Jupyter
----> 3 ssc.stop(stopSparkContext=False)

File /bigdata/spark-3.3.2-bin-hadoop3/python/pyspark/streaming/context.py:257,
in StreamingContext.stop(self, stopSparkContext, stopGraceFully)
    244 def stop(self, stopSparkContext: bool = True, stopGraceFully: bool = F
alse) -> None:
    245     """
    246     Stop the execution of the streams, with option of ensuring all
    247     received data has been processed.
   (...)
    255         data to be completed
    256     """
--> 257     self._jssc.stop(stopSparkContext, stopGraceFully)
    258     StreamingContext._activeContext = None
    259     if stopSparkContext:

File /bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4j/ja
va_gateway.py:1320, in JavaMember.__call__(self, *args)
   1313 args_command, temp_args = self._build_args(*args)
   1315 command = proto.CALL_COMMAND_NAME +\
   1316     self.command_header +\
   1317     args_command +\
   1318     proto.END_COMMAND_PART
-> 1320 answer = self.gateway_client.send_command(command)
   1321 return_value = get_return_value(
   1322     answer, self.gateway_client, self.target_id, self.name)
   1324 for temp_arg in temp_args:

File /bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4j/ja
va_gateway.py:1038, in GatewayClient.send_command(self, command, retry, binar
y)
   1036 connection = self._get_connection()
   1037 try:
-> 1038     response = connection.send_command(command)
   1039     if binary:
   1040         return response, self._create_connection_guard(connection)

File /bigdata/spark-3.3.2-bin-hadoop3/python/lib/py4j-0.10.9.5-src.zip/py4j/cl
ientserver.py:511, in ClientServerConnection.send_command(self, command)
    509 try:
    510     while True:
--> 511         answer = smart_decode(self.stream.readline()[:-1])
    512         logger.debug("Answer received: {0}".format(answer))
    513         # Happens when a the other end is dead. There might be an empt
y
    514         # answer before the socket raises an error.

File /home2/anaconda3/lib/python3.10/socket.py:705, in SocketIO.readinto(self,
b)
    703 while True:
    704     try:
--> 705         return self._sock.recv_into(b)
    706     except timeout:
    707         self._timeout_occurred = True
```

```
KeyboardInterrupt:
```

In [ ]: