Spark Streaming

Lab 01: Reading Network Streams

In this lab, we'll demonstrate streaming using a network connection. Our streaming job will wait for someone to connect to a network socket and will process the data that arrives on that socket.

Note that a new RDD is created and immediately processed each time a file is discovered.

We'll walk through the process in this lab. For this and all the streaming labs, you will require two terminal windows on your cluster. We'll qualify which instructions to type in which terminal window in each step.

Terminal 1: Starting spark-shell

First, we have to ensure we run spark with enough cores to populate one core for the Spark streaming receiver and another for a Spark DStream.

Connect to your cluster using the secure shell, and type the following to start the spark-shell with two cores:

```
spark-shell --master local[2]
```

Terminal 1: Set Up For Network Streaming

The only difference between this setup and the one in Lab 01 is that we are going to use `ssc.socketTextStream` instead of `ssc.textFileStream` to read our data. `socketTextStream` requires us to specify what socket we want to listen to, so we've chosen 999. We still will use a 10 second micro-batch interval.

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._ // not necessary since Spark 1.3

// Create a local StreamingContext with two working threads
// and a batch interval of 10 second.
// The master requires 2 cores to prevent from a starvation scenario.

val ssc = new StreamingContext(sc, Seconds(10))
val lines = ssc.socketTextStream("localhost", 9999)
```

Terminal 1: Wordcount code

Here, everything is pretty much the same as with our file-based streaming wordcount program.

```
val words = lines.flatMap(_.split(" "))

// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.print()
```

Terminal 1: Start The Stream

Now we simply start our stream using `ssc.start`.

```
ssc.start()
ssc.awaitTermination()  // Wait for the computation to terminate
```

Terminal 2: Send Some Data To Network Port 9999

We're going to use the network cat program `nc` to provide input to our stream. We do that by the following command

```
nc -lk 9999
```

The terminal will now wait for you to type words. You should type some input for the streaming program. I suggest something such as:

```
Do you like green eggs and ham?
I do not like them, Sam-I-Am.
I do not like green eggs and ham.
Would you like them here or there?
I would not like them here or there.
I would not like them anywhere.
I do not like green eggs and ham.
I do not like them, Sam-I-Am.
```

Wait for about 30 seconds after you've typed that, and then copy and paste it again.

```
Do you like green eggs and ham?
I do not like them, Sam-I-Am.
I do not like green eggs and ham.
Would you like them here or there?
I would not like them here or there.
I would not like them anywhere.
I do not like green eggs and ham.
I do not like them, Sam-I-Am.
```

Terminal 1: See The WordCount Output

You should see the words get counted on Terminal 1 soon after you add each file. It should look like the following:

```
-------------------------------------------
Time: 1504477320000 ms
-------------------------------------------
(Sam-I-Am.,2)
(here,2)
(them,,2)
(not,6)
(or,2)
(green,3)
(anywhere.,1)
(would,2)
(Would,1)
(like,8)
...

-------------------------------------------
Time: 1504477330000 ms
-------------------------------------------

17/09/03 22:22:13 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
17/09/03 22:22:13 WARN BlockManager: Block input-0-1504477333600 replicated to only 0 peer(s)
-------------------------------------------
Time: 1504477340000 ms
-------------------------------------------
(Sam-I-Am.,2)
(here,2)
(them,,2)
(not,6)
(or,2)
(green,3)
(anywhere.,1)
(would,2)
(Would,1)
(like,8)
...
```

You should now kill the job by hitting control-C in Terminal 1 and Terminal 2. You will likely see many errors on Terminal 1 as the streaming network job shuts down.

This step concludes the lab.