Spark Streaming

Review the Spark Streaming documentation

1. View the Spark Streaming API by visiting the Spark Scaladoc. API (which you bookmarked previously in the class) and selecting the org.apache.spark.streaming package in the package pane on the left.
2. Follow the links at the top of the package page to view the DStream and PairDStreamFunctions classes – these will show you the functions available on a DStream of regular RDDs and Pair RDDs respectively.
3. You may also wish to view the Spark Streaming Programming Guide (select **Programming'Guides** > **Spark'Streaming** on the Spark documentation main page).

Count Words in a Stream

For this section, you will simulate streaming text data through a network socket using the nc command. This command takes input from the console (stdin) and sends it to the port you specify, so that the text you type is sent to the client program (which will be your Spark Streaming application.)

4. In a terminal window, enter the command

```
$ nc -lkv 1234
```

   Anything you type will be sent to port 1234. You will return to this window after you have started your Spark Streaming Context.

5. Start a separate terminal for running the Spark Shell. Copy /usr/lib/spark/conf/log4j.properties to the local directory and edit it to set the logging level to ERROR. (This is to reduce the level of logging output, which otherwise would make it difficult to see the interactive output from the streaming job.)

Count Words in a Stream

6. Start the Spark Scala Shell. In order to use Spark Streaming interactively, you need to either run the shell on a Spark cluster, or locally with at least two threads. For this exercise, run locally with two threads, by typing:

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

7. In the Spark Shell, import the classes you need for this example. You may copy the commands from these instructions, or if you prefer, copy from the solution script file provided (SparkStreaming.scalaspark).

```
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.Seconds
```

Count Words in a Stream

8. Create a Spark Streaming Context, starting with the Spark Context provided by the shell, with a batch duration of 5 seconds:

```
val ssc = new StreamingContext(sc,Seconds(5))
```

9. Create a DStream to read text data from port 1234 (the same port you are sending text to using the nc command, started in the first step.)

```
val mystream = ssc.socketTextStream("localhost",1234)
```

10. Use MapReduce to count the occurrence of words on the stream.

```
val words = mystream.flatMap(line => line.split("\\W"))
val wordCounts = words.map(x =>
(x, 1)).reduceByKey((x,y) => x+y)
```

Count Words in a Stream

11. Print out the first 10 word count pairs in each batch:

```
wordCounts.print()
```

12. Start the Streaming Context. This will trigger the DStream to connect to the socket, and start batching and processing the input every 5 seconds. Call awaitTermination to wait for the task to complete.

```
ssc.start()
ssc.awaitTermination()
```

13. Go back to the terminal window in which you started the nc command. You should see a message indicating that nc accepted a connection from your DStream.

Count Words in a Stream

14. Enter some text. Every five seconds you should see output in the Spark Shell window:

```
-------------------------------------------
Time: 1396631265000 ms
-------------------------------------------
(never,1)
(purple,1)
(I,1)
(a,1)
(ve,1)
(seen,1)
```

15. To restart the application, type Ctrl-C to exit Spark Shell, then restart the shell and use command history or paste in the application commands again.
16. When you are done, close the nc process in the other terminal window.

Spark Streaming Application

Count Knowledge Base article requests

Now that you are familiar with using Spark Streaming, try a more realistic task: read in streaming web server log data, and count the number of requests for Knowledge Base articles.

To simulate a streaming data source, you will use the provided streamtest.py Python script, which waits for a connection on the host and port specified and, once it receives a connection, sends the contents of the file(s) specified to the client (which will be your Spark Streaming application). You can specify the speed at which the data should be sent (lines per second).

Writing a Spark Streaming Application

1. Stream the weblog files at a rate of 20 per second. In a separate terminal window, run

```
$ python
~/training_materials/sparkdev/examples/streamtest.py
localhost 1234 20 /training_materials/sparkdev/data/weblogs/*
```

2. Note that this script exits after the client disconnects; you will need to restart the script when you restart your Spark Application.
3. To complete the exercise, start with the stub code in src/main/scala/stubs/StreamingLogs.scala, which imports the necessary classes and sets up the Streaming Context.
4. Create a DStream by reading the data from the host and port provided as input parameters.
5. Filter the DStream to only include lines containing the string “KBDOC”.
6. For each RDD in the filtered DStream, display the number of items – that is, the number of requests for KB articles.

Writing a Spark Streaming Application

6. Save the filtered logs to text files.
7. To test your application run your application locally and be sure to specify two threads; at least two threads or nodes are required to running a streaming application, while our VM cluster has only one. The StreamingLogs application takes two parameters: the host name and port number to connect the DStream to; specify the same host and port that the test script is listening on.

```
$ spark-submit \
--class stubs.StreamingLogs \
--master local[2] \
target/streamlog-1.0.jar localhost 1234
```

Note: you may choose to use

```
--class solution.StreamingLogs
```

Writing a Spark Streaming Application

8. Verify the count output, and review the contents of the files.
9. Challenge: In addition to displaying the count every second (the batch duration), count the number of KB requests over a window of 10 seconds. Print out the updated 10 second total every 2 seconds.
   a) Hint 1: Use the countByWindow function.
   b) Hint 2: Use of window operations requires checkpointing. Use the ssc.checkpoint(*directory*) function before starting the SSC to enable checkpointing.

Broadcast Variables

Using Broadcast Variables

In this Exercise you will filter web requests to include only those from devices included in a list of target models.

The list of target models is in ~training_materials/sparkdev/data/targetmodels.txt

Filter the web server logs to include only those requests from devices in the list. (The model name of the device will be in the line in the log file.) Use a broadcast variable to pass the list of target devices to the workers that will run the filter tasks.

Hint: Use the stub file for this exercise in

~/training_materials/sparkdev/stubs for the code to load in the list of target models.

Accumulators

Using Accumulators

In this Exercise you will count the number of different types of files requested in a set of web server logs.

Using accumulators, count the number of each type of file (HTML, CSS and JPG) requested in the web server log files.

Hint: use the file extension string to determine the type of request, i.e. .html, .css, .jpg.