Lab 10- W205 Section 3 Spring 2017

Sanjay Dorairaj

# **SUBMISSION 1:** Print only words with a length > 5 characters. Submit the pyspark code

|  |
| --- |
| from pyspark import SparkContext  from pyspark.streaming import StreamingContext  ssc = StreamingContext(sc, 1)  lines= ssc.textFileStream("file:///tmp/datastreams")  uclines = lines.map(lambda word: word.upper())  uclines = uclines.filter(lambda word: len(word) > 5)  uclines.pprint()  ssc.start() |

**Output:**

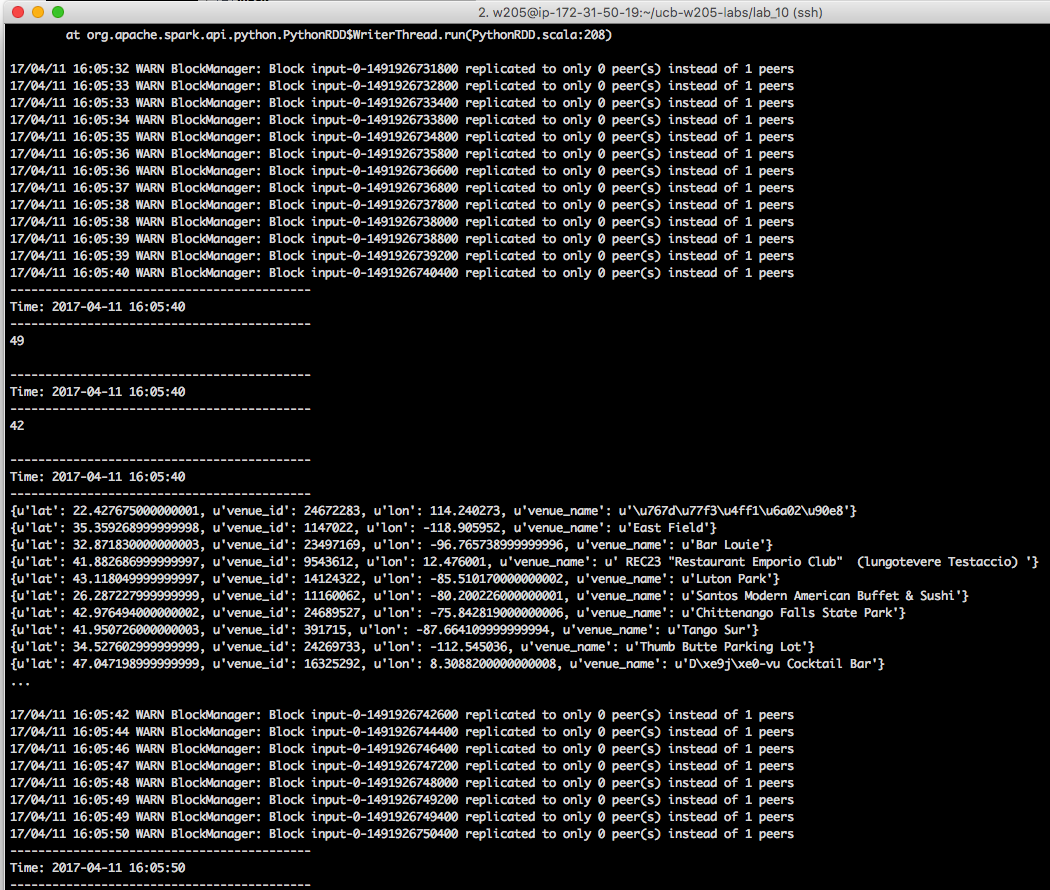
|  |
| --- |
| **-------------------------------------------**  **Time: 2017-04-11 04:20:32**  **-------------------------------------------**  **COLORADO**  **DORADO**  **ELDORADE**  **GATRORADE**  **-------------------------------------------**  **Time: 2017-04-11 04:20:33**  **-------------------------------------------**  **-------------------------------------------**  **Time: 2017-04-11 04:20:34**  **-------------------------------------------**  **-------------------------------------------**  **Time: 2017-04-11 04:20:35**  **-------------------------------------------**  **-------------------------------------------**  **Time: 2017-04-11 04:20:36**  **-------------------------------------------**  **-------------------------------------------**  **Time: 2017-04-11 04:20:37**  **-------------------------------------------**  **COLORADO**  **DORADO**  **ELDORADE**  **GATRORADE** |

# **SUBMISSION 2:** Change the code so that you save the venue components to a text file. Submit you code.

|  |
| --- |
| MASTER=local[4] pyspark  from pyspark import SparkContext  from pyspark.streaming import StreamingContext  import json  ssc = StreamingContext(sc, 10)  lines = ssc.textFileStream("file:///tmp/datastreams")  slines = lines.flatMap(lambda x: [ j['venue'] for j in json.loads('['+x+']') if 'venue' in j] )  cnt=slines.count()  cnt.pprint()  slines.pprint()  slines.saveAsTextFiles("file:///tmp/venues.txt")  ssc.start() |

# **SUBMISSION 3:** In a previous module in this class you learnt about streams, burstiness and kafka. Describe how you would solve a situation where (1) you have a very busty stream where you spark streaming process may not always be able to keep up with the data it receives, meaning it the time it takes to process takes longer than the batch interval. (2) Like other programs stream processing programs need to be updated. Describe the implications of updating this simple processing program. What side effects can it have? How could you potentially handle them?.

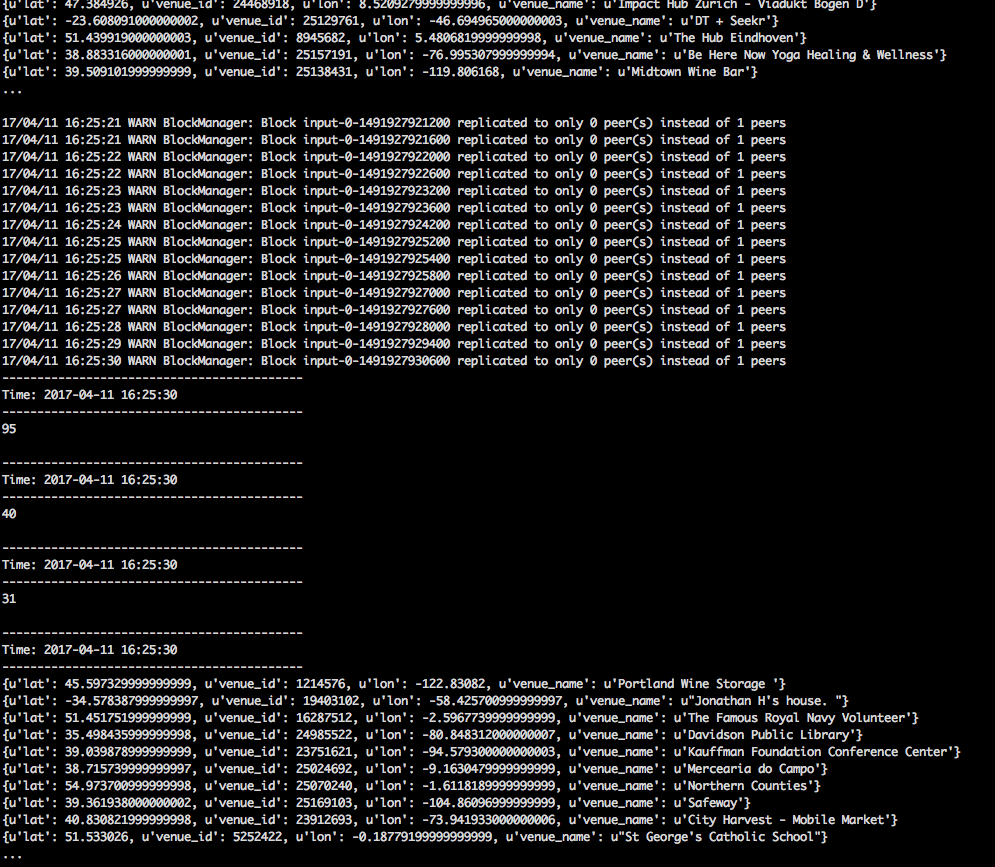
**Output with Spark Submit**



## Describe how you would solve a situation where (1) you have a very busty stream where you spark streaming process may not always be able to keep up with the data it receives, meaning it the time it takes to process takes longer than the batch interval.

1. Kafka queues have tunable parameters that allow us to adjust the retention period to ensure that programs such as Spark Streaming are able to deal with bursty traffic. A longer retention period ensures that even though there is bursty traffic, Kafka is able to buffer this bursty traffic till the stream processing program has had the time to process all the events.
2. In case, the streaming processing is unable to cope with bursty traffic despite tuning Kafka, then, the alternative would be for the streaming program to apply a filter to the incoming events so as to reduce the number of events that it processes.
3. A third alternative, is to send the events from Kafka directly to a file system and then read off that filesystem or database, although this may defeat the process of stream processing.
4. Yet another alternative could be to increase the size of the min-batches by increasing the time between consecutive processing. This may reduce transport overhead related to processing the increasing stream, giving a bit more time for the streaming program to process the incoming dstreams.

# **SUBMISSION 4a:** Provide a screenshot of the running Spark Streaming application that shows that the CountByWindow indeed provides an sum of the counts from the 3 latest batches. See example screenshot below.



# **SUBMISSION 4b:** Also explain what the difference is between having 10 sec batches with a 30 sec sliding window and a 30 second batch length.

Batch size refers to the duration over which RDDs need to be collected. If the mini-batch size is set to 10 seconds, then the dstream that is sent to the streaming program would contain RDDs for the past 10 seconds. Similarly, if the batch length is 30 seconds, then the dstream would contain RDDs for the past 30 seconds.

The window size refers to dstreams collected over a sliding window that includes prior RDDs that may have already been processed in earlier windows.

The below diagram illustrates the difference.

****

Here is a better explanation assuming the below series of events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Time**  **(base t)** | **Events**  Let’s say, some random words are being streamed | **Mini-Batch – size 10 seconds** | **Mini-Batch – size 30 seconds** | **Window – 30,10**  Window length – 30 seconds and slide interval – 10 seconds |
| T | Hello | Hello |  |  |
| T+10 | world | world |  |  |
| T+20 | This | This | Hello world This | Hello world This |
| T+30 | is | is |  | World This is |
| T+40 | a | a |  | This is a |
| T+50 | streaming | streaming | Is a streaming | Is a streaming |
| T+60 | program | program |  | A streaming program |