**Cheng, Loi**

November 4, 2019

Harvard Extension - Big Data Principles e88

Homework 9: Streaming Processing with Spark

Please identify which problems were completed. If any were incomplete, please identify where you encountered problems.

|  |
| --- |
| Problem 1: 100%  Problem 2: 100%  Problem 3: 100%  Problem 4: 100%  Bonus Problem 5: |

**Problem 1: [25 points]**

Paste most relevant source code of your Spark Streaming job into the following area.[5 points]

|  |
| --- |
| import sys  from pyspark import SparkContext, SparkConf  from pyspark.streaming import StreamingContext  from pyspark.streaming.kafka import KafkaUtils  import os  if \_\_name\_\_ == "\_\_main\_\_":      # Create Spark Context      sc = SparkContext(appName="PythonStreamingDirectKafkaCount")      ssc = StreamingContext(sc, 1)      ssc.checkpoint("checkpoint")      brokers, topic = sys.argv[1:]      print(brokers)      print(topic)      sc.setLogLevel("WARN")      #Create a DStream that will connect to Kafka      #kafkaParams = {"metadata.broker.list": brokers, "auto.offset.reset": "smallest"}      kafkaParams = {"metadata.broker.list": brokers}      kafkaStream = KafkaUtils.createDirectStream(ssc, [topic], kafkaParams)      # RDD with initial state (key, value) pairs      initialStateRDD = sc.parallelize([])      def updateFunc(new\_value, last\_sum):          return sum(new\_value) + (last\_sum or 0)      def parse\_log\_line(line):          (uuid, timestamp, url, user) = line.strip().split(":")          return (url, 1)      lines = kafkaStream.map(lambda x: x[1])      clicks = lines.map(parse\_log\_line)\          .reduceByKey(lambda a, b: a + b)      clicks.pprint()      running\_counts = clicks.updateStateByKey(updateFunc, initialRDD=initialStateRDD)      running\_counts.pprint()      # Start the computation      ssc.start()      # Wait for the computation to terminate      ssc.awaitTermination() |

Demo your Kafka producer sending events to the Kafka cluster [10 points]

|  |
| --- |
|  |

Show your Spark job's output for a few batch windows, with the results of the job - counts of the number of clicks per URL per batch and running total [10 points]

|  |
| --- |
|  |

**Problem 2: [25 points]**

Paste the most relevant source code of your job into the following area [10 points]

|  |
| --- |
| import sys  from pyspark import SparkContext, SparkConf  from pyspark.streaming import StreamingContext  from pyspark.streaming.kafka import KafkaUtils  import os  if \_\_name\_\_ == "\_\_main\_\_":      # Create Spark Context      sc = SparkContext(appName="PythonStreamingDirectKafkaCount")      ssc = StreamingContext(sc, 1)      ssc.checkpoint("checkpoint")      brokers, topic = sys.argv[1:]      print(brokers)      print(topic)      sc.setLogLevel("WARN")      #Create a DStream that will connect to Kafka      #kafkaParams = {"metadata.broker.list": brokers, "auto.offset.reset": "smallest"}      kafkaParams = {"metadata.broker.list": brokers}      kafkaStream = KafkaUtils.createDirectStream(ssc, [topic], kafkaParams)      # RDD with initial state (key, value) pairs      initialStateRDD = sc.parallelize([])      def updateFunc(new\_value, last\_sum):          return sum(new\_value) + (last\_sum or 0)      def parse\_log\_line(line):          (uuid, timestamp, url, user) = line.strip().split(":")          return (url, 1)      lines = kafkaStream.map(lambda x: x[1]).window(5,5)      clicks = lines.map(parse\_log\_line)\          .reduceByKey(lambda a, b: a + b)      clicks.pprint()      running\_counts = clicks.updateStateByKey(updateFunc, initialRDD=initialStateRDD)      running\_counts.pprint()      # Start the computation      ssc.start()      # Wait for the computation to terminate      ssc.awaitTermination() |

Show a few events that your Kafka producer is sending to the Kafka cluster [5 points]

|  |
| --- |
|  |

Show your job's console output, as well as the results of the job [10 points]

|  |
| --- |
| INFO lines were removed from the console output |

**Problem 3: [35 points]**

Paste the most relevant source code of your job [5 points]

|  |
| --- |
| import sys  from pyspark import SparkContext, SparkConf  from pyspark.streaming import StreamingContext  from pyspark.streaming.kafka import KafkaUtils  import os  if \_\_name\_\_ == "\_\_main\_\_":      # Create Spark Context      sc = SparkContext(appName="PythonStreamingDirectKafkaCount")      ssc = StreamingContext(sc, 1)      brokers, topic, accuracy = sys.argv[1:]      print(brokers)      print(topic)      sc.setLogLevel("WARN")      #Create a DStream that will connect to Kafka      #kafkaParams = {"metadata.broker.list": brokers, "auto.offset.reset": "smallest"}      kafkaParams = {"metadata.broker.list": brokers}      kafkaStream = KafkaUtils.createDirectStream(ssc, [topic], kafkaParams)      def parse\_log\_line(line):          (uuid, timestamp, url, user) = line.strip().split(":")          return (user, 1)      lines = kafkaStream.map(lambda x: x[1]).window(30,30)      def approxDistinct(time,rdd):          print("Time: %s count approx distinct..." % str(time))          try:              print('count approx distinct',rdd.countApproxDistinct(float(accuracy)))          except:              pass      #approximate      lines.foreachRDD(approxDistinct)      def countActual(time,rdd):          print("Time: %s count distinct..." % str(time))          try:              print('count distinct',rdd.count())          except:              pass      #actual      clicks = lines.map(parse\_log\_line).reduceByKey(lambda a, b: a + b)      #clicks.pprint()      clicks.foreachRDD(countActual)      # Start the computation      ssc.start()      # Wait for the computation to terminate      ssc.awaitTermination() |

Show a few events that your Kafka producer is sending to the Kafka cluster [5 points]

|  |
| --- |
|  |

Show your job's console output, as well as the results of the job - number of unique users per 30 sec windows using Spark aggregation methods [5 points]

|  |
| --- |
| See next section, the counts by aggregation and by HLL are ran together |

Show your job's console output, as well as the results of the job - number of unique users per 30 sec windows using HyperLogLog algorithm; run HLL job with a few (2-3) different accuracy settings [10 points]

|  |
| --- |
| Accuracy at 0.01    Accuracy at 0.1    Accuracy at 0.001    Accuracy at 0.5 |

Compare and explain the results of HLL with different accuracy settings and the regular aggregation results [5 points]

|  |
| --- |
| Differences between HLL and regular aggregation is close at 0.01 and 0.001 accuracy. At 0.1 accuracy the difference increases slightly. The difference is greatest at 0.5 accuracy setting |

Compare the execution flow (number of jobs) for HLL with different accuracy settings - explain the results. [5 points]

|  |
| --- |
| The execution flow (number of jobs) looks the same for difference accuracy settings. This is probably because HLL algorithm is O(1) complexity.  0.5 accuracy    0.001 accuracy |

**Problem 4: [15 points]**

Draw the state of the counters array after adding s4 element. Show all steps of your calculations [5 points]

|  |
| --- |
|  |

Draw the state of the counters array after adding s5 element. Show all steps of your calculations [5 points]

|  |
| --- |
|  |

Explain the results and what could be done to improve (if needed) them [5 points]

|  |
| --- |
| The count of s5 is 2, but we know we have only seen it once. So, there is a collision on all three hashes. We can either add another hash function h4 or increase the range of the hash function outputs to more than 4 to reduce the chance of collision on all hashes. |

**Bonus Problem 4: [+20 points]**

Paste most relevant source code of your job [10 points]

|  |
| --- |
|  |

Show a few events that your Kafka producer is sending to the Kafka cluster [2 points]

|  |
| --- |
|  |

Show results of the job - demonstrate how events are de-duped within the window, but not de-duped outside of the window [8 points]

|  |
| --- |
|  |