## HU Extension Assignment 08 E63 Big Data Analytics

### Handed out: 03/26/2016 Due by 11:30PM EST, 04/01/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used as separate files. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. Please, submit to the class drop box. For issues and comments visit the class Discussion Board. You are not obliged to use Java or Eclipse. You are welcome to use any language and any IDE of your choice.

Use most excellent and very detailed notes created by Marina Popova for Section 08 on Kafka and Streaming as you main guide for this assignment.

**Problem 1)** On your Cloudera VM or any other VM you might be using install Kafka. Just in case, install one of the recent Kafka 0.8 versions. Demonstrate that you can create a topic, publish messages to that topic and consume messages sent to that topic. Use Kafka command line interface.

I am using my CentOS 6.7 VM for this problem.

From the link <http://kafka.apache.org/downloads.html> I downloaded the file kafka\_2.11-0.8.2.2.tgz

Then I unzipped the file with the command:

[cloudera@localhost ~]$ tar -xvf kafka\_2.11-0.8.2.2.tgz

Then I created kafka-data directory:

[cloudera@localhost ~]$ pwd

/home/cloudera

[cloudera@localhost ~]$ mkdir kafka-data

So, now I have kafka directory in cloudera home directory:

[cloudera@localhost ~]$ ls kafka\*

kafka\_2.11-0.8.2.2.tgz

kafka\_2.11-0.8.2.2:

bin config libs LICENSE logs NOTICE

kafka-data:

[cloudera@localhost ~]$

Then I modified kafka’s server.properties:

[cloudera@localhost ~]$ vi kafka\_2.11-0.8.2.2/config/server.properties

And set

log.dirs=/home/cloudera/kafka-data/kafka-logs

Then I modified zooker.properties:

[cloudera@localhost ~]$ vi kafka\_2.11-0.8.2.2/config/zookeeper.properties

And set

dataDir=/home/cloudera/kafka-data/zookeeper

Then I started zookeeper:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/zookeeper-server-start.sh ~/kafka\_2.11-0.8.2.2/config/zookeeper.properties

It gave console log like:

[2016-03-31 22:43:03,859] INFO Reading configuration from: /home/cloudera/kafka\_2.11-0.8.2.2/config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2016-03-31 22:43:03,866] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)

[2016-03-31 22:43:03,866] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)

[2016-03-31 22:43:03,866] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)

[2016-03-31 22:43:03,866] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)

[2016-03-31 22:43:03,895] INFO Reading configuration from: /home/cloudera/kafka\_2.11-0.8.2.2/config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

So zookeeper was running on localhost:2181

Then I started Kafka server:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-server-start.sh ~/kafka\_2.11-0.8.2.2/config/server.properties

It gave console log like:

[2016-03-31 22:44:42,186] INFO Verifying properties (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,211] INFO Property broker.id is overridden to 0 (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,212] INFO Property log.cleaner.enable is overridden to false (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,212] INFO Property log.dirs is overridden to /home/cloudera/kafka-data/kafka-logs (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,212] INFO Property log.retention.check.interval.ms is overridden to 300000 (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,213] INFO Property log.retention.hours is overridden to 168 (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,215] INFO Property log.segment.bytes is overridden to 1073741824 (kafka.utils.VerifiableProperties)

[2016-03-31 22:44:42,215] INFO Property num.io.threads is overridden to 8 (kafka.utils.VerifiableProperties)

so then Kafka server was running on localhost:9092

Then I created kafka topic:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 4 --topic assignment8\_problem1\_topic

Created topic "assignment8\_problem1\_topic".

[cloudera@localhost ~]$

I gave the below command to describe Kafka cluster:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-topics.sh --describe --zookeeper localhost:2181

Topic:assignment8\_problem1\_topic PartitionCount:4 ReplicationFactor:1 Configs:

Topic: assignment8\_problem1\_topic Partition: 0 Leader: 0 Replicas: 0 Isr: 0

Topic: assignment8\_problem1\_topic Partition: 1 Leader: 0 Replicas: 0 Isr: 0

Topic: assignment8\_problem1\_topic Partition: 2 Leader: 0 Replicas: 0 Isr: 0

Topic: assignment8\_problem1\_topic Partition: 3 Leader: 0 Replicas: 0 Isr: 0

[cloudera@localhost ~]$

I gave the below command to list Kafka topics:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-topics.sh --list --zookeeper localhost:2181

assignment8\_problem1\_topic

[cloudera@localhost ~]$

Then I started kafka producer:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic assignment8\_problem1\_topic

[2016-03-31 23:26:55,635] WARN Property topic is not valid (kafka.utils.VerifiableProperties)

And started Kafka consumer:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic assignment8\_problem1\_topic --from-beginning

Now if I type something on Producer it appears on consumer:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic assignment8\_problem1\_topic

[2016-03-31 23:26:55,635] WARN Property topic is not valid (kafka.utils.VerifiableProperties)

hi

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic assignment8\_problem1\_topic --from-beginning

hi

I type something on producer again, it appears on consumer:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic assignment8\_problem1\_topic

[2016-03-31 23:26:55,635] WARN Property topic is not valid (kafka.utils.VerifiableProperties)

hi

hello

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic assignment8\_problem1\_topic --from-beginning

hi

hello

Then I inspected the message content:

[cloudera@localhost ~]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --files ~/kafka-data/kafka-logs/assignment8\_problem1\_topic-0/00000000000000000000.log --print-data-log

Dumping /home/cloudera/kafka-data/kafka-logs/assignment8\_problem1\_topic-0/00000000000000000000.log

Starting offset: 0

offset: 0 position: 0 isvalid: true payloadsize: 2 magic: 0 compresscodec: NoCompressionCodec crc: 4251893211 payload: hi

offset: 1 position: 28 isvalid: true payloadsize: 5 magic: 0 compresscodec: NoCompressionCodec crc: 2275900082 payload: hello

[cloudera@localhost ~]$

As you can see the messages we typed are showing up here.

Then I checked the kafka log files:

[cloudera@localhost kafka-logs]$ pwd

/home/cloudera/kafka-data/kafka-logs

[cloudera@localhost kafka-logs]$

[cloudera@localhost kafka-logs]$ ls -l assignment8\_problem1\_topic-0

total 4

-rw-rw-r--. 1 cloudera cloudera 10485760 Mar 31 23:24 00000000000000000000.index

-rw-rw-r--. 1 cloudera cloudera 59 Mar 31 23:28 00000000000000000000.log

[cloudera@localhost kafka-logs]$ ls -l assignment8\_problem1\_topic-1

total 0

-rw-rw-r--. 1 cloudera cloudera 10485760 Mar 31 23:24 00000000000000000000.index

-rw-rw-r--. 1 cloudera cloudera 0 Mar 31 23:24 00000000000000000000.log

[cloudera@localhost kafka-logs]$ ls -l assignment8\_problem1\_topic-2

total 0

-rw-rw-r--. 1 cloudera cloudera 10485760 Mar 31 23:24 00000000000000000000.index

-rw-rw-r--. 1 cloudera cloudera 0 Mar 31 23:24 00000000000000000000.log

[cloudera@localhost kafka-logs]$ ls -l assignment8\_problem1\_topic-3

total 0

-rw-rw-r--. 1 cloudera cloudera 10485760 Mar 31 23:24 00000000000000000000.index

-rw-rw-r--. 1 cloudera cloudera 0 Mar 31 23:24 00000000000000000000.log

**Problem 2)** Using Java or Python or any other (even scripting) language of your choice construct a producer and a consumer object. Let producer generate one random number between 0 or 1 and 10 every second. Let both producer and consumer run indefinitely or until you kill them. Demonstrate that your consumer is receiving messages by printing both the stream if numbers generated on the producer and the stream of numbers fetched by the consumer. You might find it easier to print to files and examine files afterwards. Once you terminate the exchange, examine Kafka’s log.

Instructions on how to write Java producer and consumer you can find on this URLs:

<https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example>

<https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example>

Instructions on how to write Python clients for Kafka you could find on this URL:

<https://cwiki.apache.org/confluence/display/KAFKA/Clients#Clients-Python>

Instructions for Scala could be found here:

<https://cwiki.apache.org/confluence/display/KAFKA/Clients#Clients-ScalaDSL>

You are welcome to follow any other instructions and use any other programming or scripting language.

For this I used JAVA language.

This is my Kafka Producer class:

**public** **class** Problem2\_Producer {

// use File Writer to output the messages

**private** **static** FileWriter *fileWriter* = **null**;

/\*\*

\* the main function of this program

\*

\* **@param** args

\* (<kafkaServerIPAndPort> <kafkaTopic>)

\*/

**public** **static** **void** main(String[] args) {

// make sure 2 arguments are passed to the program

**if** (args.length < 2) {

System.***err***.println("Usage: Problem2\_Producer <kafkaServerIPAndPort> <kafkaTopic>\n"

+ " <kafkaServerIPAndPort> is a list of one or more Kafka brokers e.g. localhost:9092\n"

+ " <kafkaTopic> is a list of one or more kafka topics to consume from e.g. assignment8\_problem2and3\_topic\n\n");

System.*exit*(1);

}

**try** {

// initialize the file writer

*fileWriter* = **new** FileWriter(**new** File("./Assignment8\_Problem2\_Producer.log"));

} **catch** (IOException e1) {

e1.printStackTrace();

System.*exit*(0);

}

// for my local instance this is "localhost:9092"

String kafkaServerIPAndPort = args[0];

// for my local instance this is "assignment8\_problem2and3\_topic"

String kafkaTopic = args[1];

// create properties map for kafka producer

Properties kafkaProducerProperties = **new** Properties();

kafkaProducerProperties.put("metadata.broker.list", kafkaServerIPAndPort);

kafkaProducerProperties.put("serializer.class", kafka.serializer.StringEncoder.**class**.getName());

// properties.put("partitioner.class"

kafkaProducerProperties.put("request.required.acks", "1");

// create kafka producer config

ProducerConfig kafkaProducerConfig = **new** ProducerConfig(kafkaProducerProperties);

// create kafka producer

Producer<String, String> kafkaProducer = **new** Producer<String, String>(kafkaProducerConfig);

// declare a variable for kafka keyed message

KeyedMessage<String, String> kafkaKeyedMessage = **null**;

// create a randm value generator

Random randomValueGenerator = **new** Random();

// log out a message saying sending kafka messages

System.***out***.println("Will start sending kafka messages to " + kafkaServerIPAndPort + " on topic:" + kafkaTopic);

**try** {

*fileWriter*.write(

"Will start sending kafka messages to " + kafkaServerIPAndPort + " on topic:" + kafkaTopic);

*fileWriter*.flush();

} **catch** (IOException e1) {

e1.printStackTrace();

}

// this is an infinite loop. so basically producer will keep messages

// until the program is shutdown

**while** (**true**) {

// get a random number between 0 and 10

**int** randomNumber = randomValueGenerator.nextInt(10);

// create an instance of keyed message

kafkaKeyedMessage = **new** KeyedMessage<String, String>(kafkaTopic, String.*valueOf*(randomNumber));

// send the message

kafkaProducer.send(kafkaKeyedMessage);

// log out the message

System.***out***.println("Kafka Producer: Sent message :" + randomNumber);

**try** {

*fileWriter*.write("\nKafka Producer: Sent message :" + randomNumber);

*fileWriter*.flush();

} **catch** (IOException e1) {

e1.printStackTrace();

System.*exit*(0);

}

**try** {

// sleep for 1 sec

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// this means the thread is interrupted by some other process.

// Exit in that case

System.*exit*(0);

}

}

}

}

This is my Kafka consumer class (this class uses Spark DStreams):

public class Problem2\_Consumer\_DStream {

public static void main(String[] args) {

if (args.length < 3) {

System.err.println("Usage: Problem2\_Consumer\_DStream <kafkaBrokers> <topics> <zookeeper>\n"

+ " <brokers> is a list of one or more Kafka brokers e.g. localhost:9092\n"

+ " <topics> is a list of one or more kafka topics to consume from e.g. assignment8\_problem2and3\_topic\n"

+ " <zookeeper> is the IP and port on which server is running e.g. localhost:2181\n");

System.exit(1);

}

// for my local instance this is "localhost:9092";

String kafkaBrokers = args[0];

// for my local instance this is assignment8\_problem2and3\_topic

String kafkaTopic = args[1];

// for my local instance this is localhost:2181

String zookeeper = args[2];

// Create a Java Spark Config.

SparkConf sparkConf = new SparkConf().setAppName("Assignment8\_Problem3\_Consumer");

// sparkConf.setMaster("local[5]"); // this is to run the program as a

// standalone java application

// create spark context from the spark configuration

JavaSparkContext sparkContext = new JavaSparkContext(sparkConf);

// create spark streaming context with batch interval of 1 second

JavaStreamingContext streamingContext = new JavaStreamingContext(sparkContext, Durations.seconds(1));

// create a kafka topics set

HashSet<String> kafkaTopicsSet = new HashSet<String>(Arrays.asList(kafkaTopic.split(",")));

// create a map for kafka input DStream params

HashMap<String, String> kafkaInputDStreamParams = new HashMap<String, String>();

kafkaInputDStreamParams.put("metadata.broker.list", kafkaBrokers);

kafkaInputDStreamParams.put("zookeeper.connect", zookeeper);

kafkaInputDStreamParams.put("group.id", "Assignment8\_Problem3\_Consumer");

System.out.println("Listening for kafka messages from " + kafkaBrokers + " on topic:" + kafkaTopic);

// create a pair input DStream

JavaPairInputDStream<String, String> pairInputDStream = KafkaUtils.createDirectStream(streamingContext,

String.class, String.class, StringDecoder.class, StringDecoder.class, kafkaInputDStreamParams,

kafkaTopicsSet);

// create a stream of numbers received as messages

JavaDStream<String> numbersReceivedAsMessages = pairInputDStream

.map(new Function<Tuple2<String, String>, String>() {

private static final long serialVersionUID = 1L;

@Override

public String call(Tuple2<String, String> tuple2) {

return tuple2.\_2();

}

});

// print the messages to output

numbersReceivedAsMessages.print();

// Start the computation

streamingContext.start();

streamingContext.awaitTermination();

}

}

Note: This class uses Spark DStream class.

I have also submitted a file called Problem2\_Consumer\_KafkaAPI.java in the deliverables which uses **kafka.javaapi.consumer.SimpleConsumer** as mentioned in the link <https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example>

Both my Kafa producer and consumer classes use following dependencies through maven:

<dependencies>

<dependency> <!-- Spark dependency -->

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.10</artifactId>

<version>1.6.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_2.10</artifactId>

<version>1.6.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming-kafka\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.19</version>

</dependency>

</dependencies>

Both the classes are in different projects.

I exported jar files using maven for both the projects using the commands:

[cloudera@localhost Assignment8\_Problem2\_Producer]$ mvn clean && mvn compile && mvn package

and

[cloudera@localhost Assignment8\_Problem2\_Consumer\_DStream]$ mvn clean && mvn compile && mvn package

Then I submitted spark job for Producer:

[cloudera@localhost assignment8\_input\_output]$ spark-submit --class e63.kafka.streaming.Problem2\_Producer Assignment8\_Problem2\_Producer.jar localhost:9092 assignment8\_problem2and3\_topic

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

Will start sending kafka messages to localhost:9092 on topic:assignment8\_problem2and3\_topic

Time: 04:53:26 Kafka Producer: Sent message :6

Time: 04:53:27 Kafka Producer: Sent message :2

Time: 04:53:28 Kafka Producer: Sent message :5

Time: 04:53:29 Kafka Producer: Sent message :2

Time: 04:53:30 Kafka Producer: Sent message :1

Time: 04:53:31 Kafka Producer: Sent message :8

Time: 04:53:32 Kafka Producer: Sent message :2

Time: 04:53:33 Kafka Producer: Sent message :3

Time: 04:53:34 Kafka Producer: Sent message :7

I then submitted spark jobs for Consumer program:

[cloudera@localhost assignment8\_input\_output]$ spark-submit --class e63.kafka.streaming.Problem2\_Consumer\_DStream Assignment8\_Problem2\_Consumer\_DStream.jar localhost:9092 assignment8\_problem2and3\_topic localhost:2181

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

16/04/01 16:53:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/04/01 16:53:21 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.71.158 instead (on interface eth0)

16/04/01 16:53:21 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

16/04/01 16:53:24 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.

Listening for kafka messages from localhost:9092 on topic:assignment8\_problem2and3\_topic

-------------------------------------------

Time: 1459554807000 ms

-------------------------------------------

6

-------------------------------------------

Time: 1459554808000 ms

-------------------------------------------

2

-------------------------------------------

Time: 1459554809000 ms

-------------------------------------------

5

-------------------------------------------

Time: 1459554810000 ms

-------------------------------------------

2

-------------------------------------------

Time: 1459554811000 ms

-------------------------------------------

1

-------------------------------------------

Time: 1459554812000 ms

-------------------------------------------

8

-------------------------------------------

Time: 1459554813000 ms

-------------------------------------------

2

As you can see messages sent by producer are received by consumer

Deliverables:

* Problem2\_Consumer\_DStream.java (consumer for problem2 that uses spark DStream)
* Problem2\_Consumer\_KafkaAPI.java (consumer for problem2 that uses Kafka SimpleConsumer API)
* Problem2\_Producer.java (producer for problem2)
* pom.xml (maven config file. Changed to pom.xml.txt for submission)
* producer\_when\_running\_problem2.log.txt (log file output when I ran Problem2\_Producer.java in order to test problem 2)
* consumer\_with\_KafkaAPI\_when\_running\_roblem2.log.txt (log file output when I ran Problem2\_Consumer\_KafkaAPI.java in order to test problem 2)

**Problem 3)** Starting from one of the attached Spark Streaming clients DirectKafkaWordCount in Java, Scala or Python write a consumer client that will replace the consumer from the previous problem. However, rather than simply printing every message it receives from the producer, let it print for us every 5 seconds the rolling count of numbers between 1 and 10 it received in the last 30 seconds. You might find it simpler to print to files and then examine those files afterwards. For Java build simple Maven Project with a single Java class and pom.xml file similar to the one provided. Build your projects following the process we used in Assignment 4.

You are welcome to follow any other instructions and use any other programming or scripting language to accomplish the above goals.

For this problem I used JAVA.

This is my program:

**public** **class** Problem3\_Consumer {

// initialize the logger that is based on slf4j library

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(Problem3\_Consumer.**class**);

/\*\*

\* The main function for this class

\*

\* **@param** args

\* (<brokers> <topics>)

\*/

**public** **static** **void** main(String[] args) {

// make sure 2 arguments are passed to the program

**if** (args.length < 3) {

System.***err***.println("Usage: Problem3\_Consumer <kafkaBrokers> <topics> <zookeeper>\n"

+ " <brokers> is a list of one or more Kafka brokers e.g. localhost:9092\n"

+ " <topics> is a list of one or more kafka topics to consume from e.g. assignment8\_problem2and3\_topic\n"

+ " <zookeeper> is the IP and port on which server is running e.g. localhost:2181\n");

System.*exit*(1);

}

// for my local instance this is "localhost:9092";

String kafkaBrokers = args[0];

// for my local instance this is assignment8\_problem2and3\_topic

String kafkaTopic = args[1];

// for my local instance this is localhost:2181

String zookeeper = args[2];

// Create a Java Spark Config.

SparkConf sparkConf = **new** SparkConf().setAppName("Assignment8\_Problem3\_Consumer");

sparkConf.setMaster("local[5]"); // this is to run the program as a

// standalone java application

// create spark context from the spark configuration

JavaSparkContext sparkContext = **new** JavaSparkContext(sparkConf);

// create spark streaming context with batch interval of 1 second

JavaStreamingContext streamingContext = **new** JavaStreamingContext(sparkContext, Durations.*seconds*(1));

// create a kafka topics set

HashSet<String> kafkaTopicsSet = **new** HashSet<String>(Arrays.*asList*(kafkaTopic.split(",")));

// create a map for kafka input DStream params

HashMap<String, String> kafkaInputDStreamParams = **new** HashMap<String, String>();

kafkaInputDStreamParams.put("metadata.broker.list", kafkaBrokers);

kafkaInputDStreamParams.put("zookeeper.connect", zookeeper);

kafkaInputDStreamParams.put("group.id", "Assignment8\_Problem3\_Consumer");

***LOGGER***.info("Listening for kafka messages from " + kafkaBrokers + " on topic:" + kafkaTopic);

// create a pair input DStream

JavaPairInputDStream<String, String> pairInputDStream = KafkaUtils.*createDirectStream*(streamingContext,

String.**class**, String.**class**, StringDecoder.**class**, StringDecoder.**class**, kafkaInputDStreamParams,

kafkaTopicsSet);

// create a stream of numbers received as messages

JavaDStream<String> numbersReceivedAsMessages = pairInputDStream

.map(**new** Function<Tuple2<String, String>, String>() {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Override

**public** String call(Tuple2<String, String> tuple2) {

**return** tuple2.\_2();

}

});

// create a pair DStream of numbers received as messages and their

// count.

// where the number received (in the form of string) will be key

// and the count of number will be the value

JavaPairDStream<String, Integer> numberAndCountAsOnePairs = numbersReceivedAsMessages

.mapToPair(**new** PairFunction<String, String, Integer>() {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Override

**public** Tuple2<String, Integer> call(String s) {

**return** **new** Tuple2<String, Integer>(s, 1);

}

});

// create a reduce function that will add the counts of numbers received

Function2<Integer, Integer, Integer> reduceFunction = **new** Function2<Integer, Integer, Integer>() {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Override

**public** Integer call(Integer i1, Integer i2) **throws** Exception {

**return** i1 + i2;

}

};

// here reduceByKeyAndWindow is used to create a sliding windowed

// stream.

// the window duration is 30 seconds

// and slide interval is 5 seconds

JavaPairDStream<String, Integer> numberAndCountPairs = numberAndCountAsOnePairs

.reduceByKeyAndWindow(reduceFunction, Durations.*seconds*(30), Durations.*seconds*(5));

// call functionToLogOutEachRDD on each RDD of numberAndCountPairs so

// that the RDDs can be printed out in logs

numberAndCountPairs.print();

// Start the computation and await termination

streamingContext.start();

streamingContext.awaitTermination();

}

}

The program is in a project called ‘Assignment8\_Problem3’

The pom.xml for that project has following dependencies:

<dependencies>

<dependency> <!-- Spark dependency -->

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.10</artifactId>

<version>1.6.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_2.10</artifactId>

<version>1.6.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming-kafka\_2.10</artifactId>

<version>1.6.0</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.19</version>

</dependency>

</dependencies>

I exported the jar file for this project as:

[cloudera@localhost Assignment8\_Problem3\_Consumer]$ mvn clean && mvn compile && mvn package

I submitted the program to spark in the following way:

[cloudera@localhost assignment8\_input\_output]$ spark-submit --class e63.kafka.streaming.Problem3\_Consumer Assignment8\_Problem3\_Consumer.jar localhost:9092 assignment8\_kafka\_topic localhost:2181

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

16/04/01 00:22:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/04/01 00:22:10 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.71.158 instead (on interface eth0)

16/04/01 00:22:10 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Running the programs:

This is the sample output of Producer from Problem 2

[cloudera@localhost assignment8\_input\_output]$ spark-submit --class e63.kafka.streaming.Problem2\_Producer Assignment8\_Problem2\_Producer.jar localhost:9092 assignment8\_problem2and3\_topic

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

Will start sending kafka messages to localhost:9092 on topic:assignment8\_problem2and3\_topic

Time: 05:03:20 Kafka Producer: Sent message :0

Time: 05:03:21 Kafka Producer: Sent message :6

Time: 05:03:22 Kafka Producer: Sent message :3

Time: 05:03:23 Kafka Producer: Sent message :2

Time: 05:03:24 Kafka Producer: Sent message :8

Time: 05:03:25 Kafka Producer: Sent message :7

Time: 05:03:26 Kafka Producer: Sent message :0

Time: 05:03:27 Kafka Producer: Sent message :3

Time: 05:03:28 Kafka Producer: Sent message :8

Time: 05:03:29 Kafka Producer: Sent message :8

Time: 05:03:30 Kafka Producer: Sent message :4

Time: 05:03:31 Kafka Producer: Sent message :5

Time: 05:03:32 Kafka Producer: Sent message :6

Time: 05:03:33 Kafka Producer: Sent message :2

Time: 05:03:34 Kafka Producer: Sent message :8

Time: 05:03:35 Kafka Producer: Sent message :7

Time: 05:03:36 Kafka Producer: Sent message :1

Time: 05:03:37 Kafka Producer: Sent message :9

Time: 05:03:38 Kafka Producer: Sent message :3

Time: 05:03:39 Kafka Producer: Sent message :9

Time: 05:03:40 Kafka Producer: Sent message :9

Time: 05:03:41 Kafka Producer: Sent message :0

Time: 05:03:42 Kafka Producer: Sent message :0

Time: 05:03:43 Kafka Producer: Sent message :3

Time: 05:03:44 Kafka Producer: Sent message :8

Time: 05:03:45 Kafka Producer: Sent message :3

Time: 05:03:46 Kafka Producer: Sent message :6

Time: 05:03:47 Kafka Producer: Sent message :3

Time: 05:03:48 Kafka Producer: Sent message :3

Time: 05:03:49 Kafka Producer: Sent message :5

Time: 05:03:50 Kafka Producer: Sent message :0

Time: 05:03:51 Kafka Producer: Sent message :0

Time: 05:03:52 Kafka Producer: Sent message :6

Time: 05:03:53 Kafka Producer: Sent message :5

Time: 05:03:54 Kafka Producer: Sent message :2

Time: 05:03:55 Kafka Producer: Sent message :4

Time: 05:03:56 Kafka Producer: Sent message :3

Time: 05:03:57 Kafka Producer: Sent message :1

Time: 05:03:58 Kafka Producer: Sent message :0

Time: 05:03:59 Kafka Producer: Sent message :4

Time: 05:04:00 Kafka Producer: Sent message :1

Time: 05:04:01 Kafka Producer: Sent message :9

and corresponding sample output of Consumer from Problem 3:

[cloudera@localhost assignment8\_input\_output]$ spark-submit --class e63.kafka.streaming.Problem3\_Consumer Assignment8\_Problem3\_Consumer.jar localhost:9092 assignment8\_problem2and3\_topic localhost:2181

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

16/04/01 17:03:14 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

16/04/01 17:03:15 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using 192.168.71.158 instead (on interface eth0)

16/04/01 17:03:15 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

16/04/01 17:03:17 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.

-------------------------------------------

Time: 1459555403000 ms

-------------------------------------------

(3,1)

(0,1)

(6,1)

-------------------------------------------

Time: 1459555408000 ms

-------------------------------------------

(7,1)

(2,1)

(8,1)

(3,2)

(0,2)

(6,1)

-------------------------------------------

Time: 1459555413000 ms

-------------------------------------------

(7,1)

(2,1)

(8,3)

(3,2)

(4,1)

(5,1)

(0,2)

(6,2)

-------------------------------------------

Time: 1459555418000 ms

-------------------------------------------

(7,2)

(2,2)

(8,4)

(3,2)

(4,1)

(9,1)

(5,1)

(0,2)

(6,2)

(1,1)

-------------------------------------------

Time: 1459555423000 ms

-------------------------------------------

(7,2)

(2,2)

(8,4)

(3,3)

(4,1)

(9,3)

(5,1)

(0,4)

(6,2)

(1,1)

-------------------------------------------

Time: 1459555428000 ms

-------------------------------------------

(7,2)

(2,2)

(8,5)

(3,6)

(4,1)

(9,3)

(5,1)

(0,4)

(6,3)

(1,1)

-------------------------------------------

Time: 1459555433000 ms

-------------------------------------------

(7,2)

(2,2)

(8,5)

(3,6)

(4,1)

(9,3)

(5,2)

(0,5)

(6,3)

(1,1)

-------------------------------------------

Time: 1459555438000 ms

-------------------------------------------

(7,1)

(2,2)

(8,4)

(3,6)

(4,2)

(9,3)

(5,3)

(0,4)

(6,3)

(1,2)

-------------------------------------------

Time: 1459555443000 ms

-------------------------------------------

(7,1)

(2,2)

(8,2)

(3,6)

(4,2)

(9,4)

(5,2)

(0,5)

(6,2)

(1,3)

-------------------------------------------

Time: 1459555448000 ms

-------------------------------------------

(2,1)

(8,1)

(3,6)

(4,2)

(9,3)

(5,2)

(0,5)

(6,2)

(1,2)

-------------------------------------------

Time: 1459555453000 ms

-------------------------------------------

(2,1)

(8,1)

(3,5)

(4,2)

(9,1)

(5,2)

(0,3)

(6,2)

(1,2)

-------------------------------------------

Time: 1459555458000 ms

-------------------------------------------

(2,1)

(3,2)

(4,2)

(9,1)

(5,2)

(0,3)

(6,1)

(1,2)

-------------------------------------------

Time: 1459555463000 ms

-------------------------------------------

(2,1)

(3,1)

(4,2)

(9,1)

(5,1)

(0,1)

(1,2)

-------------------------------------------

Time: 1459555468000 ms

-------------------------------------------

(4,1)

(9,1)

(0,1)

(1,1)

-------------------------------------------

As you can see **window operation is performed after every 5 mins**.

And the **window interval during this operation is last 30 mins**

I later made a slight modification to the code so that it prints output to the log file:

VoidFunction<JavaPairRDD<String, Integer>> functionToLogOutEachRDD = **new** VoidFunction<JavaPairRDD<String, Integer>>() {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Override

**public** **void** call(JavaPairRDD<String, Integer> javaPairRDD) **throws** Exception {

// some calendar and date operations to print time stamps for

Calendar calendar = Calendar.*getInstance*();

calendar.add(Calendar.***SECOND***, -30);

Date currentTimeStamp = **new** Date();

Date timeStampOf30SecondsBefore = calendar.getTime();

***LOGGER***.info("");

***LOGGER***.info("Current time: " + ***dateTimeFormat***.format(currentTimeStamp)

+ " showing count of numbers in last 30 secs (from "

+ ***dateTimeFormat***.format(timeStampOf30SecondsBefore) + " to now)");

// for each functinon to log out each tuple of number and count

javaPairRDD.foreach(functionToLogOutEachTuple);

}

// this function is to access each tuple in the RDD and then print

// that tuple to the log wih LOGGER

VoidFunction<Tuple2<String, Integer>> functionToLogOutEachTuple = **new** VoidFunction<Tuple2<String, Integer>>() {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Override

**public** **void** call(Tuple2<String, Integer> tuple) **throws** Exception {

***LOGGER***.info("Number:" + tuple.\_1() + " count:" + tuple.\_2());

}

};

};

// call functionToLogOutEachRDD on each RDD of numberAndCountPairs so

// that the RDDs can be printed out in logs

numberAndCountPairs.foreachRDD(functionToLogOutEachRDD);

I have attached log output of this modified consumer with assignment deliverables.

This is the dump output of kafka logs after running problem 2 and problem 3 (both those problems use the same kafka topic (assignment8\_problem2and3\_topic):

[cloudera@localhost kafka-logs]$ ~/kafka\_2.11-0.8.2.2/bin/kafka-run-class.sh kafka.tools.DumpLogSegments --files ~/kafka-data/kafka-logs/assignment8\_problem2and3\_topic-0/00000000000000000000.log --print-data-log

Dumping /home/cloudera/kafka-data/kafka-logs/assignment8\_problem2and3\_topic-0/00000000000000000000.log

Starting offset: 0

offset: 0 position: 0 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 879748084 payload: 9

offset: 1 position: 27 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1256103753 payload: 4

offset: 2 position: 54 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3568962282 payload: 3

offset: 3 position: 81 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3568962282 payload: 3

offset: 4 position: 108 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 2747087484 payload: 2

offset: 5 position: 135 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1256103753 payload: 4

offset: 6 position: 162 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1130943330 payload: 8

offset: 7 position: 189 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3568962282 payload: 3

offset: 8 position: 216 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 879748084 payload: 9

offset: 9 position: 243 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 2765160037 payload: 6

offset: 10 position: 270 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1130943330 payload: 8

offset: 11 position: 297 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3568962282 payload: 3

offset: 12 position: 324 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1256103753 payload: 4

offset: 13 position: 351 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1256103753 payload: 4

offset: 14 position: 378 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 2747087484 payload: 2

offset: 15 position: 405 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1037676511 payload: 5

offset: 16 position: 432 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1130943330 payload: 8

offset: 17 position: 459 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1037676511 payload: 5

offset: 18 position: 486 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1256103753 payload: 4

offset: 19 position: 513 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1037676511 payload: 5

offset: 20 position: 540 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1130943330 payload: 8

offset: 21 position: 567 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1130943330 payload: 8

offset: 22 position: 594 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1037676511 payload: 5

offset: 23 position: 621 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 879748084 payload: 9

offset: 24 position: 648 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 984902598 payload: 1

offset: 25 position: 675 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3568962282 payload: 3

offset: 26 position: 702 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 1130943330 payload: 8

offset: 27 position: 729 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3554135795 payload: 7

offset: 28 position: 756 isvalid: true payloadsize: 1 magic: 0 compresscodec: NoCompressionCodec crc: 3554135795 payload: 7

[cloudera@localhost kafka-logs]$

Deliverables:

* Problem3\_Consumer.java (the java file with sliding window kafka consumer required for Problem 3)
* producer\_when\_running\_problem3.log.txt (log file output when I ran Problem2\_Producer.java in order to test problem 3)
* sliding\_window\_consumer\_problem3.log.txt (log file output when I ran Problem3\_Consumer.java in order to test problem 3)