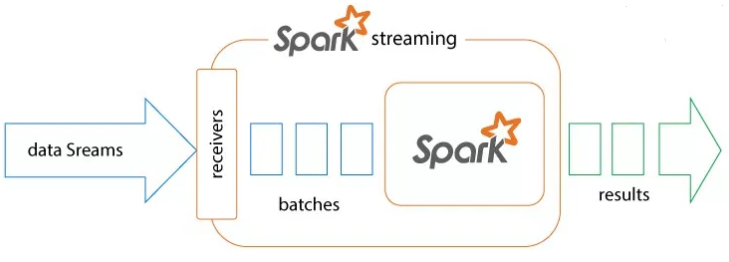
Session 22

Assignment 2

|  |  |
| --- | --- |
| **Prepared For:** | AcadGild |
|  |  |
| **Document Approval:** | **AcadGild** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Project Title:** | Session 22 – Assignment 2 |
|  |  |
| **Prepared By:** | Duncan Burgess |
|  |  |
|  | dburgess@duncb.com |
|  |  |
| **Primary Engineer:** | Duncan Burgess |
|  |  |
| **Document Reference:** | **Session 21 – Assignment 2** |
|  |  |
| **Start Date:** | 22/10/2017 |
|  |  |
|  |  |



# 

# Contents

[Contents 2](#_Toc496435443)

[Change History 3](#_Toc496435444)

[1. Problem Statement 4](#_Toc496435445)

[2. Stateful Streaming in Apache Spark 4](#_Toc496435446)

[2.1. Preperation 5](#_Toc496435447)

[2.2. Code written 5](#_Toc496435448)

[2.3. Creation of the Kafka environment in Windows 7](#_Toc496435449)

[2.4. Hadoop 8](#_Toc496435450)

[2.5. Results 8](#_Toc496435451)

[2.6. Results 9](#_Toc496435452)

# Change History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Document Revision** | **Date** | **Authored By** | **Authorised By** | **Sections Affected** | **Reason for Change** |
| Rev 01 | 22/10/2017 | Duncan Burgess |  | All | Initial release. |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Problem Statement

Follow the following blog

# Stateful Streaming in Apache Spark

Apache Spark is a general processing engine built on top of the Hadoop eco-system. Spark has a complete setup and a unified framework to process any kind of data. Spark can do batch processing as well as stream processing. Spark has a powerful SQL engine to run SQL queries on the data; it also has an integrated Machine Learning library called MlLib and a graph processing library called GraphX. As it can integrate many things into it, we identify Spark as a unified framework rather than a processing engine.

Now coming to the real-time stream processing engine of Spark. Spark doesn’t process the data in real time it does a near-real-time processing. It means it processes the data in micro batches, in just a few milliseconds.

Here we have got a program where Spark’s streaming context will process the data in micro batches but generally, this processing is stateless. Let’s take we have defined the streaming Context to run for every 10 seconds, it will process the data that is arrived within that 10 seconds, to process the previous data we have something called windows concept, windows cannot give the accumulated results from the starting timestamp.

But what if you need to the accumulate the results from the start of the streaming job. Which means you need to check the previous state of the RDD in order to update the new state of the RDD. This is what is known as stateful streaming in Spark.

Spark provides 2 API’s to perform stateful streaming, which is **updateStateByKey** and **mapWithState**.

Now we will see how to perform stateful streaming of wordcount using **updateStateByKey**. UpdateStateByKey is a function of Dstreams in Spark which accepts an update function as its parameter. In that update function, you need to provide the following parameters **newState** for the key which is a **seq of values** and the previous state of key as an Option[?].

Let’s take a word count program, let’s say for the first 10 seconds we have given this data:

This is a message

Duncan Duncan Duncan

Now the wordcount program result will be

(This,1)

(is,1)

(a,1)

(message,1)

(Duncan,3)

Now without writing the updateStateByKey function, if you give some other data, in the next 10 seconds i.e. let’s assume we give the same line hello every one from acadigld. Now we will get the same result in the next 10 seconds

also i.e.,

(This,1)

(is,1)

(a,1)

(message,1)

(Duncan,3)

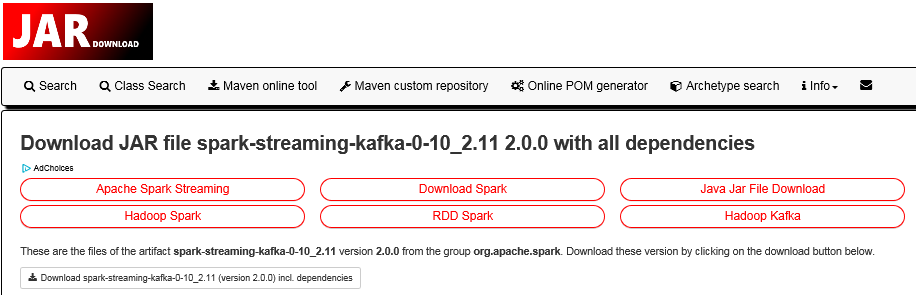
Now, what if we need an accumulated result of the wordcount which counts my previous results also. This is where stateful streaming comes into the act. In stateful streaming, your key’s previous state will be preserved and it will be updated with new results.

Note: For performing stateful operations, you will need a key value pair because streamingContext remembers the state of your RDD based on the keys itself.

In our previous blog on Kafka-Spark-Streaming integration, we have discussed about how to integrate Apache spark with Kafka and do realtime processing. We recommend our users to go through our previous blog on Kafka Spark integration to generate your input to the Spark streaming job using Kafka-producer console.

## Preperation

Downloaded the required jars



## Code written

**package** com.duncb.spark

**import** org.apache.spark.{ SparkConf, SparkContext }

**import** org.apache.spark.streaming.StreamingContext

**import** org.apache.spark.streaming.Seconds

**import** org.apache.spark.streaming.dstream.DStream

**import** org.apache.spark.rdd.RDD

**import** org.apache.spark.streaming.{ State, StateSpec }

**import** org.apache.spark.streaming.kafka010.KafkaUtils

**import** org.apache.kafka.common.serialization.StringDeserializer

**import** org.apache.kafka.clients.consumer.ConsumerRecord

**import** org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent

**import** org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

**object** stateFulWordCount {

**def** main(args: Array[*String*]) {

**val** conf = **new** SparkConf().setMaster("local[\*]").setAppName("KafkaReceiver")

**val** ssc = **new** StreamingContext(conf, Seconds(10))

/\*

\* Defining the Kafka server parameters

\*/

**val** kafkaParams = Map[*String*, Object](

"bootstrap.servers" -> "localhost:9092,localhost:9092",

"key.deserializer" -> classOf[StringDeserializer],

"value.deserializer" -> classOf[StringDeserializer],

"group.id" -> "use\_a\_separate\_group\_id\_for\_each\_stream",

"auto.offset.reset" -> "latest",

"enable.auto.commit" -> (**false**: java.lang.Boolean))

**val** topics = Array("acadgild-topic") //topics list

**val** kafkaStream = KafkaUtils.createDirectStream[*String*, *String*](

ssc,

PreferConsistent,

Subscribe[*String*, *String*](topics, kafkaParams))

**val** splits = kafkaStream.map(record => (record.key(), record.value.toString)).flatMap(x => x.\_2.split(" "))

**val** updateFunc = (values: *Seq*[Int], state: Option[Int]) => {

**val** currentCount = values.foldLeft(0)(\_ + \_)

**val** previousCount = state.getOrElse(0)

**Some**(currentCount + previousCount)

}

//Defining a check point directory for performing stateful operations

ssc.checkpoint("hdfs://10.100.100.77:9000/WordCount\_checkpoint")

**val** wordCounts = splits.map(x => (x, 1)).reduceByKey(\_+\_).updateStateByKey(updateFunc)

kafkaStream.print() //prints the stream of data received

wordCounts.print() //prints the wordcount result of the stream

ssc.start()

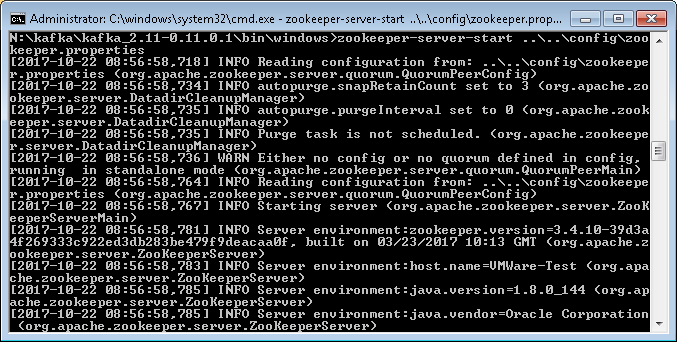
ssc.awaitTermination()

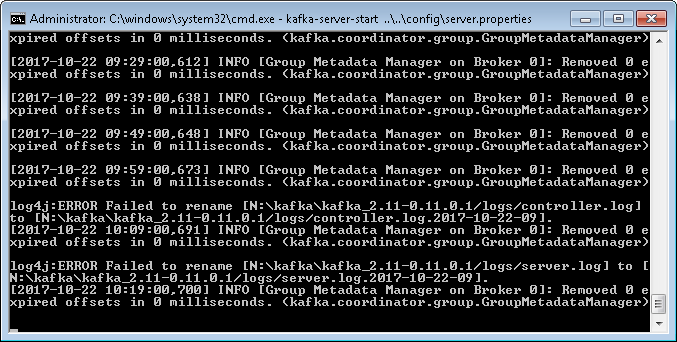
}

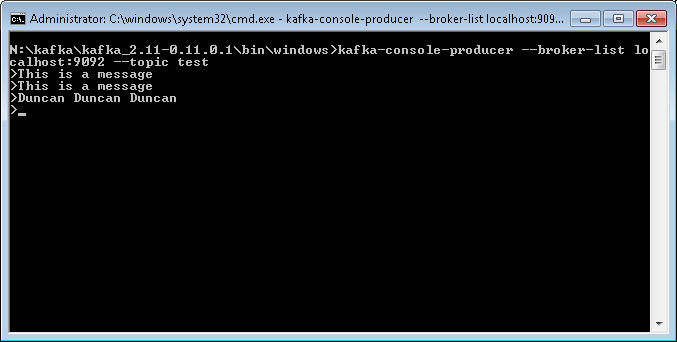
}

## Creation of the Kafka environment in Windows

Installation of zookeeper and Kafka plus console







## Hadoop

The updateFunc will work on each key, for every key in the RDD, this update function will run, it will take the last state of your key and it will check for the new values for your and the data operation whatever you want to do for your key and return the new values as a Some().

For working with this update function, you need to mandatorily provide a Checkpoint directory for your SparkStreamingContext as

ssc.checkpoint(“hdfs://10.100.100.77:9000/WordCount\_checkpoint”)

Because your intermediate values will be stored in this checkpoint directory for fault tolerance, it is suggested that you give your checkpoint directory in HDFS for more fault tolerance.

In the above update function, we are getting the new values of that key as a Seq[Int] and the oldValues of that key as Option[Int](Which is already calculated). Now inside the function, we aggregating the newValues of the key using the foldLeft function and then we are getting the old state value of the key and we are adding the both to the Some() and returning the updated sum of the values.

## Results

Results

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/N:/Spark2/jars/slf4j-log4j12-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/N:/kafka/kafka\_2.11-0.11.0.1/libs/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/N:/kafkajar/slf4j-log4j12-1.7.21.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]

The expected result where not produced I have checked all components

* Kafka server working fine and can connect.
* Can connect to the Hadoop cluster on port 9000.

Even so I have understood the theory and the workings of stateful streaming.

I will contact the help desk to see if there is a resolution to this issue. acad