ACADGILD

BIG DATA HADOOP & SPARK TRAINING

SESSION – 22

Assignment - **22.2**

TABLE OF CONTENTS

Problem Statement:

Implement the below blog at your end and send the complete documentation.

https://acadgild.com/blog/stateful-streaming-in-spark/

Introduction:

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.

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, but generally, this processing is stateless.

In this application, 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 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.**

In this application, 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 **hello every one.** Now the wordcount program result will be:
(one,1)
(hello,1)
(every,1)

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.** Now we will get the same result in the next 10 seconds also i.e.
(one,1)
(hello,1)
(every,1)

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.

Code:

```
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.spli

    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://localhost:9000/WordCount_checkpoint")

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

    ssc.start()
    ssc.awaitTermination()

  }

}
```

Building the application with SBT:

Below is the Spark Streaming and Kafka dependencies which are needed to build the applicatio n with SBT.

build.sbt

```
name := "stateFulWordCount"

version := "0.1"

scalaVersion := "2.11.8"

libraryDependencies += "org.apache.spark" % "spark-streaming_2.11" % "2.1.0"

libraryDependencies += "org.apache.spark" % "spark-core_2.11" % "2.1.0"

libraryDependencies += "org.apache.spark" % "spark-streaming-kafka-0-10_2.11" % "2.1.0"

libraryDependencies += "org.apache.kafka" % "kafka_2.11" % "0.10.2.0"

libraryDependencies += "org.apache.kafka" % "kafka-clients" % "0.10.2.0"
```

Below is the creation of the application using SBT and Eclipse. (**stateFulWordCount**)

acadgild@localhost:~/StatefulStreaming

File Edit View Search Terminal Help

```
[acadgild@localhost ~]$ cd StatefulStreaming
[acadgild@localhost StatefulStreaming]$
[acadgild@localhost StatefulStreaming]$ sbt eclipse
[info] Loading global plugins from /home/acadgild/.sbt/0.13/plugins
[info] Set current project to stateFulWordCount (in build file:/home/acadgild/StatefulStreaming/)
[info] About to create Eclipse project files for your project(s).
[info] Successfully created Eclipse project files for project(s):
[info] stateFulWordCount
[acadgild@localhost StatefulStreaming]$
```