Web: Inceptez.com Mail: info@inceptez.com Call: 7871299810, 7871299817

SPARK STREAMING

Usecase 1:

File Streams: For reading data from files on any file system compatible with the HDFS API, a DStream can be created and manipulated in runtime in the given sequence of interval-

Step1:
Goto package – org.inceptez.streaming

Step2:
Create an object - filestream

Step3:
Replace whole content with the below code

```
package org.inceptez.streaming

import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.sql.SparkSession

object filestream {

 def main(args:Array[String])
 {

// Create the context
// val sparkConf = new SparkConf().setAppName("textstream").setMaster("local[*]")
//val sparkcontext = new SparkContext(sparkConf)
val sparkSession =
SparkSession.builder.appName("textstream").enableHiveSupport.master("local[*]").getOrCreate();
val sparkcontext = sparkSession.sparkContext;

sparkcontext.setLogLevel("ERROR")

val ssc = new StreamingContext(sparkcontext, Seconds(10))

val lines = ssc.textFileStream("file:///home/hduser/sparkdata/streaming/")
val courses = lines.flatMap(_.split(" "))
val coursesCounts = courses.map(x => (x, 1)).reduceByKey(_ + _)
coursesCounts.print()
ssc.start()
ssc.awaitTermination()
}
}
```

Step4:

Execute the below command in another linux terminal to create file stream in hdfs/linux.

```
cp /home/hduser/mrdata/courses.log /home/hduser/sparkdata/streaming/c1
cp /home/hduser/mrdata/courses.log /home/hduser/sparkdata/streaming/c2
cp /home/hduser/mrdata/courses.log /home/hduser/sparkdata/streaming/c3
```

Usecase 2:

Network Socket Streams:

This DStream represents the stream of data that will be received from the data server. Each record in this DStream is a line of text.

Create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

Step1:
Create a package – org.inceptez.streaming

Step2:
Create an object - socketstream

Step3:
Replace whole content with the below code

```
package org.inceptez.pkg
import org.apache.spark.SparkConf
import  org.apache.spark.SparkContext
import org.apache.spark.streaming.{Seconds, StreamingContext}
import StreamingContext._
import org.apache.spark.SparkContext._
import java.lang.Math
object socketstream1 {

 def main(args:Array[String])

 {

//val sparkConf = new SparkConf().setAppName("auctionsocketstream").setMaster("local[2]")
//val sparkcontext = new SparkContext(sparkConf)

val sparkSession =
SparkSession.builder.appName("auctionsocketstream").enableHiveSupport.master("local[*]").getOrCreate();
val sparkcontext = sparkSession.sparkContext;

sparkcontext.setLogLevel("ERROR")

//index names
val auctionid = 0
val bid = 1
val bidtime = 2
val bidder = 3
val bidderrate = 4
val openbid = 5
val price = 6
val itemtype = 7
val daystolive = 8

   // Create the context
val ssc = new StreamingContext(sparkcontext, Seconds(20))
val lines = ssc.socketTextStream("localhost", 9999)
val auctionRDD = lines.map(line => line.split("~"))

val items_auctionRDD = auctionRDD.map(x => (x(itemtype), 1)).reduceByKey((x, y) => x + y)

   //Identify which item has more auction response

 items_auctionRDD.print()

   //total number of items (auctions)

 val totalitems = auctionRDD.map(line => line(0)).count()
totalitems.print()
items_auctionRDD.saveAsTextFiles("hdfs://localhost:54310/user/hduser/auctionout/auctiondata")

   ssc.start()
   ssc.awaitTermination()
   //ssc.awaitTerminationOrTimeout(50000)
  }
}
```

Step4:

Open the netcat socket and start paste the below content multiple times:

```
nc -lk 9999
```

```
3406945791~30~0.02539~shanie713~0~9.99~232.5~palm~7
3406945791~52~0.0575~jvross524~-1~9.99~232.5~palm~7
3406945791~95~0.66248~yebo~441~9.99~232.5~ssd~7
3406945791~125~0.66262~yebo~441~9.99~232.5~mmc~7
3406945791~145~0.6628~yebo~441~9.99~232.5~tab~7
3406945791~152.5~0.66304~yebo~441~9.99~232.5~laptop~7
3406945791~175~0.76547~lilbitreading~2~9.99~232.5~laptop~7
3406945791~195~0.80953~jaguarhw~1~9.99~232.5~pc~7
3406945791~195~0.93341~sirvinsky~52~9.99~232.5~mmc~7
3406945791~200~0.93351~sirvinsky~52~9.99~232.5~palm~7
3406945791~200~1.10953~jaguarhw~1~9.99~232.5~palm~7
3406945791~215~3.68168~lilbitreading~2~9.99~232.5~palm~7
3406945791~205~4.04531~jaguarhw~1~9.99~232.5~palm~7
3406945791~225~4.04568~jaguarhw~1~9.99~232.5~palm~7
3406945791~220~6.60786~robb1069~3~9.99~232.5~palm~7
3406945791~225~6.60799~robb1069~3~9.99~232.5~palm~7
3406945791~230~6.60815~robb1069~3~9.99~232.5~palm~7
3406945791~230~6.67638~jaguarhw~1~9.99~232.5~palm~7
3406945791~232.5~6.67674~jaguarhw~1~9.99~232.5~palm~7
```

Usecase 3:

Kafka Streams:

This kafka consumer DStream represents the stream of data that will be received from multiple sources pushed to kafka topics directly or using tools like NIFI. Each record in this DStream is a line of text.
Create a createDirectStream kafka consumer Dstream that represents streaming data from a Kafka queue, specified with topic, broker list etc.

Step1:
Goto package – org.inceptez.streaming

Step2:
Create an object - kafkastream

Step3:
Replace whole content with the below code

```
package org.inceptez.streaming
import org.apache.spark.SparkConf
import  org.apache.spark.SparkContext
import org.apache.spark.streaming.{Seconds, StreamingContext}
import StreamingContext._
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

object kafkastream {

 def main(args:Array[String])
 {
val sparkConf = new SparkConf().setAppName("kafkastream").setMaster("local[*]")
val sparkcontext = new SparkContext(sparkConf)
val ssc = new StreamingContext(sparkcontext, Seconds(10))

//ssc.checkpoint("checkpointdir")
val kafkaParams = Map[String, Object](
"bootstrap.servers" -> "localhost:9092",
      "key.deserializer" -> classOf[StringDeserializer],
      "value.deserializer" -> classOf[StringDeserializer],
      "group.id" -> "kafkatest1",
      "auto.offset.reset" -> "earliest"       )

    val topics = Array("tk1")
```

```
       val stream = KafkaUtils.createDirectStream[String, String](ssc,
        PreferConsistent,
        Subscribe[String, String](topics, kafkaParams)
       )

val kafkastream = stream.map(record => (record.key, record.value))
val inputStream = kafkastream.map(rec => rec._2);
val words = inputStream.flatMap(_.split(" "))
val results = words.map(x => (x, 1L)).reduceByKey(_ + _)
inputStream.print();
results.saveAsTextFiles("hdfs://localhost:54310/user/hduser/kafkastreamout/outdir")
ssc.start()
ssc.awaitTermination()
 } }
```

Step4:

Check if zookeeper and kafka are running, if not use the below commands to start it and start the producer to push some random space delimited data.

Start the Zookeeper Coordination service:

```
zookeeper-server-start.sh -daemon /usr/local/kafka/config/zookeeper.properties
```

Start the Kafka server:

```
kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties
```

```
kafka-console-producer.sh --broker-list localhost:9092 --topic tk1
```

Keyin some messages

Additional Usecase:

Execute the file streaming method with the integration of dataframe, DSL and Declarative queries in one program.

```
package org.inceptez.streaming
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.sql.SparkSession

object bidsfilestream {

case class Auction(auctionid: String, bid: Float, bidtime: Float, bidder: String, bidderrate:
Integer, openbid: Float, price: Float, item: String, daystolive: Integer)

def main(args:Array[String])
{

//Old method
  //val sparkConf = new SparkConf().setAppName("textstream").setMaster("local[*]")
  //val sparkcontext = sparkSession.SparkContext(sparkConf)

  // Create the context

  val sparkSession =
SparkSession.builder.appName("textstream").enableHiveSupport.master("local[*]").getOrCreate
();
val sparkcontext = sparkSession.sparkContext;
//val sqlcontext = sparkSession.sqlContext;

sparkcontext.setLogLevel("ERROR")

val ssc = new StreamingContext(sparkcontext, Seconds(10))

//val auctionRDD = sparkcontext.textFile("file:///home/hduser/sparkdata/streaming/");

val auctionRDD = ssc.textFileStream("file:///home/hduser/sparkdata/streaming/")

// create an RDD of Auction objects

// change ebay RDD of Auction objects to a DataFrame
import sparkSession.implicits._
```

```
// Foreach rdd function is an iterator on the streaming micro batch rdds as like map function

auctionRDD.foreachRDD(rdd => {

 if(!rdd.isEmpty)
        {

  val ebay = rdd.map(_.split("~")).map(p => Auction(p(0), p(1).toFloat, p(2).toFloat,p(3),
p(4).toInt, p(5).toFloat, p(6).toFloat, p(7), p(8).toInt))

  val ebaydf = ebay.toDF;

  ebaydf.createOrReplaceTempView("ebayview");

  val auctioncnt = sparkSession.sql("select count(1) from ebayview");

  print("executing table query\n")

  print("total auctions are\n")

  auctioncnt.show(1,false);


  print("Executing dataframe\n")

  print("total distinct auctions\n")


  val count = ebaydf.select("auctionid").distinct.count

  System.out.println(count);


  print("Executing dataframe\n")

  print("Max bid, sum of price of items are\n")


  import org.apache.spark.sql.functions._

  val grpbid =
ebaydf.groupBy("item").agg(max("bid").alias("max_bid"),sum("price").alias("sp")).sort($"sp"
.desc)

  grpbid.show(10,false);
```

```
        }
})

print("streaming executing in x seconds\n")

ssc.start()
ssc.awaitTermination()
}
}
```