‹ Back to Table of Contents (../index.html)

Streaming Word Count

This is a *hello world* example of Spark Streaming which counts words on 1 second batches of streaming data.

It uses an in-memory string generator as a dummy source for streaming data.

Configurations

Configurations that control the streaming app in the notebook

```
// === Configuration to control the flow of the application ===
val stopActiveContext = true
// "true"  = stop if any existing StreamingContext is running;
// "false" = dont stop, and let it run undisturbed, but your latest code may
not be used

// === Configurations for Spark Streaming ===
val batchIntervalSeconds = 1
val eventsPerSecond = 1000    // For the dummy source

// Verify that the attached Spark cluster is 1.4.0+
require(sc.version >= "1.4.0", "Spark 1.4.0+ is required to run this notebook.
Please attach it to a Spark 1.4.0+ cluster.")
```

```
stopActiveContext: Boolean = true
batchIntervalSeconds: Int = 1
eventsPerSecond: Int = 1000
```

Imports

Import all the necessary libraries. If you see any error here, you have to make sure that you have attached the necessary libraries to the attached cluster.

In this particular notebook, make sure you have attached Maven dependencies `spark-streaming-kinesis` for right version of Spark and corresponding `kinesis-client` library.

```
import org.apache.spark._
import org.apache.spark.storage._
import org.apache.spark.streaming._
```

```
import org.apache.spark._
import org.apache.spark.storage._
import org.apache.spark.streaming._
```

Setup: Define the function that sets up the StreamingContext

In this we will do two things.

- Define a custom receiver as the dummy source (no need to understand this)
- Define the function that creates and sets up the streaming computation (this is the main logic)

```
// This is the dummy source implemented as a custom receiver. No need to
understand this.

import scala.util.Random
import org.apache.spark.streaming.receiver._

class DummySource(ratePerSec: Int) extends Receiver[String]
(StorageLevel.MEMORY_AND_DISK_2) {

  def onStart() {
    // Start the thread that receives data over a connection
    new Thread("Dummy Source") {
      override def run() { receive() }
    }.start()
  }

  def onStop() {
   // There is nothing much to do as the thread calling receive()
   // is designed to stop by itself isStopped() returns false
  }

  /** Create a socket connection and receive data until receiver is stopped */
  private def receive() {
    while(!isStopped()) {
      store("I am a dummy source " + Random.nextInt(10))
      Thread.sleep((1000.toDouble / ratePerSec).toInt)
    }
  }
}

import scala.util.Random
import org.apache.spark.streaming.receiver._
defined class DummySource
```

```
var newContextCreated = false      // Flag to detect whether new context was
created or not

// Function to create a new StreamingContext and set it up
def creatingFunc(): StreamingContext = {

  // Create a StreamingContext
  val ssc = new StreamingContext(sc, Seconds(batchIntervalSeconds))

  // Create a stream that generates 1000 lines per second
  val stream = ssc.receiverStream(new DummySource(eventsPerSecond))

  // Split the lines into words, and then do word count
  val wordStream = stream.flatMap { _.split(" ")  }
  val wordCountStream = wordStream.map(word => (word, 1)).reduceByKey(_ + _)

  // Create temp table at every batch interval
  wordCountStream.foreachRDD { rdd =>
    rdd.toDF("word", "count").registerTempTable("batch_word_count")
  }

  stream.foreachRDD { rdd =>
    System.out.println("# events = " + rdd.count())
    System.out.println("\t " + rdd.take(10).mkString(", ") + ", ...")
  }

  ssc.remember(Minutes(1))  // To make sure data is not deleted by the time we
query it interactively

  println("Creating function called to create new StreamingContext")
  newContextCreated = true
  ssc
}
```

```
newContextCreated: Boolean = false
creatingFunc: ()org.apache.spark.streaming.StreamingContext
```

Start Streaming Job: Stop existing StreamingContext if any and start/restart the new one

Here we are going to use the configurations at the top of the notebook to decide whether to stop any existing StreamingContext, and start a new one, or recover one from existing checkpoints.

```
// Stop any existing StreamingContext
if (stopActiveContext) {
  StreamingContext.getActive.foreach { _.stop(stopSparkContext = false) }
}

// Get or create a streaming context
val ssc = StreamingContext.getActiveOrCreate(creatingFunc)
if (newContextCreated) {
  println("New context created from currently defined creating function")
} else {
  println("Existing context running or recovered from checkpoint, may not be
running currently defined creating function")
}

// Start the streaming context in the background.
ssc.start()

// This is to ensure that we wait for some time before the background streaming
job starts. This will put this cell on hold for 5 times the
batchIntervalSeconds.
ssc.awaitTerminationOrTimeout(batchIntervalSeconds * 5 * 1000)
```

```
Creating function called to create new StreamingContext
New context created from currently defined creating function
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.St
reamingContext@2c2ebd8b
res1: Boolean = false
```

Interactive Querying

Now let's try querying the table. You can run this command again and again, you will find the numbers changing.

```
%sql select * from batch_word_count
```

word
4

8
am
0
dummy
5
a

Finally, if you want stop the StreamingContext, you can uncomment and execute the following

```
StreamingContext.getActive.foreach { _.stop(stopSparkContext = false) }
```