Case Study – IV
Spark Streaming

1. There are two parts this case study

First Part –

You have to create a Spark Application which streams data from a file on local directory on your machine and does the word count on the fly. The word should be done by the spark application in such a way that as soon as you drop the file in your local directory, your spark application should immediately do the word count for you.

```
// usage within spark-shell:
hdfsWordCount.main(Array("hdfs://quickstart.acadgild:8020/user/acadgild/sparkStreaming/"))

import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import StreamingContext._
import org.apache.hadoop.conf._
import org.apache.hadoop.fs._


/**
 * Counts words in new text files created in the given directory
 * Usage: HdfsWordCount <directory>
 *   <directory> is the directory that Spark Streaming will use to find and read new text files.
 *
 * To run this on your local machine on directory `localdir`, run this example
 *    $ bin/run-example \
 *      org.apache.spark.examples.streaming.HdfsWordCount localdir
 *
 * Then create a text file in `localdir` and the words in the file will get counted.
 */
object HdfsWordCount {
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: HdfsWordCount <directory>")
      System.exit(1)
    }

    //StreamingExamples.setStreamingLogLevels()
    val sparkConf = new SparkConf().setAppName("HdfsWordCount")
    // Create the context
    val ssc = new StreamingContext(sparkConf, Seconds(2))

    // Create the FileInputDStream on the directory and use the
    // stream to count words in new files created
    val lines = ssc.textFileStream(args(0))
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

Create an HDFS directory “/user/acadgild/sparkStreaming” where you will add your incoming files hadoop fs -mkdir /user/acadgild/sparkStreaming)

Then, from spark-shell run the program.

```
HdfsWordCount.main(Array('hdfs://quickstart.acadgild:8020/user/acadgild/sparkStreaming/'))

HdfsWordCount.main(Array('hdfs:///user/acadgild/sparkStreaming/'))
```

Add a file into /user/acadgild/sparkStreaming. hadoop fs -put <localsrc> ... <HDFS_dest_Path>

Once you have added a file into the HDFS directory, you should see in the spark shell the words of the file you just added being counted. As illustration, here is what I got after adding some LICENSE file.

```
#!/bin/sh
exec scala "$0" "$@"
!#
# ---
# Here the scala code above
# ---
HdfsWordCount.main(Array('hdfs://quickstart.acadgild:8020/user/acadgild/sparkStreaming/'))
```

Second Part –

In this part, you will have to create a Spark Application which should do the following

1. Pick up a file from the local directory and do the word count
2. Then in the same Spark Application, write the code to put the same file on HDFS.
3. Then in same Spark Application, do the word count of the file copied on HDFS in step 2
4. Lastly, compare the word count of step 1 and 2. Both should match, other throw an error

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
object SparkWordCount {
 def main(args: Array[String]) {
 // create Spark context with Spark configuration
 val sc = new SparkContext(new SparkConf().setAppName("Spark Count"))
 // get threshold
 val threshold = args(1).toInt
 // read in text file and split each document into words
 val tokenized = sc.textFile(args(0)).flatMap(_.split(" "))
 // count the occurrence of each word
 val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)
Spark Guide | 9
Developing Spark Applications
 // filter out words with fewer than threshold occurrences
 val filtered = wordCounts.filter(_._2 >= threshold)
 // count characters
 val charCounts = filtered.flatMap(_._1.toCharArray).map((_, 1)).reduceByKey(_ + _)
 System.out.println(charCounts.collect().mkString(", "))
 }
}
```

To compile Scala, include the Scala tools plug-in:

```
<plugin>
 <groupId>org.scala-tools</groupId>
 <artifactId>maven-scala-plugin</artifactId>
 <executions>
 <execution>
 <goals>
 <goal>compile</goal>
 <goal>testCompile</goal>
 </goals>
 </execution>
 </executions>
</plugin>
```

which requires the scala-tools plug-in repository:

```
<pluginRepositories>
```

```
<pluginRepository>
 <id>scala-tools.org</id>
 <name>Scala-tools Maven2 Repository</name>
 <url>http://scala-tools.org/repo-releases</url>
 </pluginRepository>
</pluginRepositories>
```

Also, include Scala and Spark as dependencies:

```
<dependencies>
 <dependency>
 <groupId>org.scala-lang</groupId>
 <artifactId>scala-library</artifactId>
 <version>2.10.2</version>
 <scope>provided</scope>
 </dependency>
 <dependency>
 <groupId>org.apache.spark</groupId>
 <artifactId>spark-core_2.10</artifactId>
 <version>1.6.0-cdh5.7.0</version>
 <scope>provided</scope>
 </dependency>
</dependencies>
```

To generate the application JAR, run:

```
$ mvn package

$ wget --no-check-certificate .../inputfile.txt
$ hdfs dfs -put inputfile.txt

$ spark-submit --class com.acadgild.sparkwordcount.SparkWordCount \
--master local --deploy-mode client --executor-memory 1g \
--name wordcount --conf "spark.app.id=wordcount" \
sparkwordcount-1.0-SNAPSHOT-jar-with-dependencies.jar
hdfs://namenode_host:8020/path/to/inputfile.txt
```