Assignment 5 - Spark Streaming

1. Write a standalone application in Spark to count the number of words in a received stream.

    - **Hint 1:** use the *netcat* server to send the text through a TCP connection. For example, `nc -lk 9999` can be used to transfer any text that you type in the terminal through port 9999.
    - **Hint 2:** if you are using `local` as the master URL when creating `StreamingContext`, you have to give at least one more core as the number of input streams, because each input stream would create a receiver that occupies one core. If you use `local`, it only gives one core to the context, which is used by the socket stream's receiver, leaving no core available for processing the data, thus, you should use `local[2]`.

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel

object NetworkWordCount {
  def main(args: Array[String]) {
    val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
    val lines = ssc.socketTextStream("127.0.0.1", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(x => (x, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}

// simple.sbt:
name := "Stream Word Count"

version := "1.0"

scalaVersion := "2.10.3"

libraryDependencies ++= Seq(
    "org.apache.spark" %% "spark-core" % "0.9.0-incubating",
    "org.apache.spark" %% "spark-streaming" % "0.9.0-incubating"
)

resolvers += "Akka Repository" at "http://repo.akka.io/releases/"

// open two terminals, and run the following command on the first terminal
$ nc -lk 9999

// then compile and run the application on the second terminal
$ sbt package
$ sbt run
```

2. Extend the code to generate word count over last 30 seconds of data, and repeat the computation every 10 seconds.

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel

object NetworkWordCount {
  def main(args: Array[String]) {
    val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
    val lines = ssc.socketTextStream("127.0.0.1", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val windowedWordCounts = pairs.window(Seconds(30), Seconds(10)).reduceByKey(_ + _)
    windowedWordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

3. Maintain a continuously updated word count for all the words in the stream.

```scala
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel

object NetworkWordCount {
  def main(args: Array[String]) {
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }

    val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
    ssc.checkpoint(".")
    val lines = ssc.socketTextStream("127.0.0.1", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val stateWordCounts = pairs.updateStateByKey[Int](updateFunc)
    stateWordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```