Assignment 3 - Spark

1. Launch the Spark shell.

```
spark-shell
```

2. Make a parallel collection of `Array(1, 2, 3, 4, 5)` and sum up all its elements.

```
sc.parallelize(Array(1, 2, 3, 4, 5)).reduce(_ + _)
```

3. Create an RDD named `pagecounts` from the given input file *hamlet*.

```
val pagecounts = sc.textFile("hamlet")
```

4. Get the first 10 lines of *hamlet* (i.e., first 10 records of `pagecounts`).

```
pagecounts.take(10)
```

5. Make a more readable print of the step 4.

```
pagecounts.take(10).foreach(println)
```

6. Count the total records in the data set `pagecounts`, and confirm its correctness by comparing the result with the Bash `wc` command: `wc -l hamlet`.

```
pagecounts.count
```

7. Monitor the jobs through the web interface.
   Spark web console to see the progress: http://127.0.0.1:4040 (Note that this page is only available if you have an active job or Spark shell). Spark Standalone cluster status web interface: http://127.0.0.1:8080.

8. Filter the data set `pagecounts` and return the items that have the word *this*.

```
val linesWithThis = pagecounts.filter(line => line.contains("this"))
```

9. Cache the new data set in memory, to avoid reading from disks.

```
val linesWithThis = pagecounts.filter(line => line.contains("this")).cache
```

10. Find the lines with the most number of words.

```
linesWithThis.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
//or
import java.lang.Math
linesWithThis.map(line => line.split(" ").size).reduce((a, b) => Math.max(a, b))
```

11. Count the total number words.

```
val wordCounts = linesWithThis.flatMap(line => line.split(" ")).count
//or
val wordCounts = linesWithThis.flatMap(_.split(" ")).count
```

12. Count the number of unique words.

```
val uniqueWordCounts = linesWithThis.flatMap(_.split(" ")).distinct.count
```

13. Count the number of each word.

```
val eachWordCounts = linesWithThis.flatMap(_.split(" ")).map(word => (word, 1))
  .reduceByKey((a, b) => a + b)
//or
val eachWordCounts = linesWithThis.flatMap(_.split(" ")).map(word => (word, 1))
  .reduceByKey(_ + _)
```

14. Save the data set in a text file.

```
eachWordCounts.saveAsTextFile("word_count")
```

15. Collect the word counts in the shell.

```
eachWordCounts.collect
```

16. Write a standalone application in Spark to count the total number of words in the input file *hamlet*.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

object WordCount {
  def main(args: Array[String]) {
    val logFile = "hamlet"
    val sc = new SparkContext("local", "Word Count", "/opt/spark-0.9.1",
      List("target/scala-2.10/word-count_2.10-1.0.jar"))
    val logData = sc.textFile(logFile, 2).cache()
    val wordCounts = logData.flatMap(line => line.split(" ")).map(word => (word, 1))
      .reduceByKey((a, b) => a + b)
    wordCounts.foreach(println(_))
    }
}

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

version := "1.0"

scalaVersion := "2.10.3"

libraryDependencies += "org.apache.spark" %% "spark-core" % "0.9.0-incubating"

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

// compile and run the application
$ sbt package
$ sbt run
```

17. Follow the instructions in the given source codes and run the word count applications on MapReduce and Stratosphere.
See the instructions in the given source code.