Spark

Use scala-shell for all questions except for the question 9. For the question 9, use spark-submit command to execute the Scala class. Provide code comments wherever applicable.

1. Consider two RDD dataset: RDD1 has number 1 to 10, RDD2 has number 5 to 10
Combine the two dataset, remove duplicate and find maximum number.
Hint: Use distint

```
val input1= sc.parallelize(List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
val input2= sc.parallelize(List(1, 2, 3, 4, 5))        // create datasets
val combine = input1.union (input2)                    //combine the two datasets
val result = combine.distinct()                        //remove duplicate
val max = result.max()                                 //find maximun number
```

result: 10

```
scala> val input1= sc.parallelize(List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
input1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize
at <console>:24

scala> val input2= sc.parallelize(List(1, 2, 3, 4, 5))
input2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at parallelize
at <console>:24

scala> val combine = input1.union (input2)
combine: org.apache.spark.rdd.RDD[Int] = UnionRDD[2] at union at <console>:27

scala> val result = combine.distinct()
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[5] at distinct at <cons
ole>:25
```

Output:

```
scala> result.collect().foreach{ println}
4
6
8
10
2
1
3
7
9
5

scala> val max = result.max()
max: Int = 10
```

2. Read a text file into a RDD. Filter and show all the sentences where the line **does not** contain a word as “ERROR”. Count number of lines which **does not** contain a word as “ERROR.

Hint: Use filter. For count see the Spark class notes “with Accumulator” example.

Input:
/tmp/task2.txt:
ERROR HUI
ERROR QI
CORRECT FEI
CORRECT QIAN

```
val input = sc.textFile("/tmp/task2.txt")                          // create dataset from txt file
val linesWithoutERROR = input.filter(line => !(line.contains("ERROR")))   // filter
linesWithoutERROR.collect().foreach{ println}                      // print specific sentences
println(linesWithoutERROR.count())                                 // count numebr
```

Result:
CORRECT FEI
CORRECT QIAN

```
scala> val input = sc.textFile("/tmp/task2.txt")
input: org.apache.spark.rdd.RDD[String] = /tmp/task2.txt MapPartitionsRDD[7] at
textFile at <console>:24

scala> val linesWithoutERROR = input.filter(line => !(line.contains("ERROR")))
linesWithoutERROR: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[8] at fil
ter at <console>:25

scala> linesWithoutERROR.collect().foreach{ println}
CORRECT FEI
CORRECT QIAN

scala> println(linesWithoutERROR.count())
2
```

3. Use the RDD operation to break a word into letters. Each letter should be separated by comma as delimiter in the output. Print only the first 5 characters. **Hint:** use flatMap

Input:
/tmp/task3.txt:
SPARKCLASS

```
val input = sc.textFile("/tmp/task3.txt")
val letterList = input.flatMap(x => x)                          // map a word to letters
val result = letterList.collect().take(5).mkString(",")         // make string with first 5 characters separated by
                                                                //comma as delimiter
```

Result:
S,P,A,R,K

```
scala> val input = sc.textFile("/tmp/task3.txt")
input: org.apache.spark.rdd.RDD[String] = /tmp/task3.txt MapPartitionsRDD[10] at
 textFile at <console>:24

scala> val letterList = input.flatMap(x => x)
letterList: org.apache.spark.rdd.RDD[Char] = MapPartitionsRDD[11] at flatMap at
<console>:25

scala> val result = letterList.collect().take(5).mkString(",")
result: String = S,P,A,R,K
```

4. Read a file file /tmp/person.txt contains firstname lastname as showsn below:
Xi Jinping
Hu Jintao
Xi Zemin
Hu Sangkun

Load the entire file into RDD. Count how many person with same first names. For example, how many Xi and how many Hu are there.

Hint: Load first name last name into key value pairs. Group by first name.

Input:
/tmp/person.txt

```
val person = sc.textFile("/tmp/person.txt")
val namePair = person.map(x => (x.split(" ")(0), x.split(" ")(1)))    //make key-value pairs (firstname, lastname)
val  firstName = namePair.keys                                        // create RDD with all keys
val result = firstName.flatMap(x => x.split(" ")).countByValue()   // group by key
```

Result:
Xi -> 2
Hu -> 2

```
scala> val person = sc.textFile("/tmp/person.txt")
person: org.apache.spark.rdd.RDD[String] = /tmp/person.txt MapPartitionsRDD[24]
at textFile at <console>:24

scala> val namePair = person.map(x => (x.split(" ")(0),x.split(" ")(1)))
namePair: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[25] at m
ap at <console>:25

scala> val firstName = namePair.keys
firstName: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[26] at keys at <c
onsole>:25

scala> firstName.collect().foreach{println}
Xi
Hu
Xi
Hu

scala> val result = firstName.flatMap(x => x.split(" ")).countByValue()
result: scala.collection.Map[String,Long] = Map(Xi -> 2, Hu -> 2)

scala>
```

5. Take the above input file /tmp/person.txt. Sort the file based on the first name. Print only the first record after sorting.

Input:
/tmp/person.txt

```
val person = sc.textFile("/tmp/person.txt")
val namePair = person.map(x => (x.split(" ")(0), x))        // make key-value pairs (firstname, fullname)
val personSort = namePair.sortByKey()                       // Sort pairs by firstname
personSort.values.collect().take(1).foreach{println}        // Print fullname of first one
```

Result:
Hu Jintao

Output:

```
scala> val namePair = person.map(x => (x.split(" ")(0), x))
namePair: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[46] at m
ap at <console>:25

scala> val personSort = namePair.sortByKey()
personSort: org.apache.spark.rdd.RDD[(String, String)] = ShuffledRDD[47] at sort
ByKey at <console>:25

scala> personSort.values.collect().take(1).foreach{println}
Hu Jintao
```

6. Take the above input file /tmp/person.txt. Convert all the first names converted to uppercase and print only first names (do not print last names).

`

Input: /tmp/person.txt

```
val person = sc.textFile("/tmp/person.txt")
val upperPerson = person.map(x => (x.split(" ")(0).toUpperCase,x.split(" ")(1)))
// make key-value pairs (firstname in uppercase, fullname)
upperPerson.keys.collect().foreach{println}
```

Output:
XI
HU
XI
HU

```
scala> val upperPerson = person.map(x => (x.split(" ")(0).toUpperCase,x.split("
")(1)))
upperPerson: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[63] a
t map at <console>:25

scala> upperPerson.keys.collect().foreach{println}
XI
HU
XI
HU
```

7. Take the above input file /tmp/person.txt. Search for first name Xi and print all people with same first name Xi and search result would be Xi Jinping and Xi Zemin.
Hint: Use lookup

Input: /tmp/person.txt

```
val person = sc.textFile("/tmp/person.txt")
val namePair = person.map(x => (x.split(" ")(0),x))
val Xi = namePair.lookup("Xi")                    // query the records with firstname == “Xi”
Xi.foreach{println}
```

Output:
Xi Jinping
Xi Zemin

```
scala> val Xi = namePair.lookup("Xi")
Xi: Seq[String] = WrappedArray(Xi Jinping, Xi Zemin)

scala> Xi.foreach{println}
Xi Jinping
Xi Zemin
```

8. Take the above input file /tmp/person.txt. Also, the second input file /tmp/person1.txt as follows:

Mao Zedong
Xi Zedong

Load two text files into two RDDs. Show the person with common first name in both the files (i.e. result is **Xi**).

Input:
/tmp/person.txt, /tmp/person1.txt

```
val person = sc.textFile("/tmp/person.txt")
val person1 = sc.textFile("/tmp/person1.txt")
val namePair = person.map(x => (x.split(" ")(0),x))
val namePair1 = person1.map(x => (x.split(" ")(0),x))
val firstName = namePair.keys
val firstName1 = namePair1.keys
firstName.intersection(firstName1).collect.foreach(println)
// Show the person with common first name in both RDD
```

Output:
Xi

```
scala> val person1 = sc.textFile("/tmp/person1.txt")
person1: org.apache.spark.rdd.RDD[String] = /tmp/person1.txt MapPartitionsRDD[52
] at textFile at <console>:24

scala> val namePair1 = person1.map(x => (x.split(" ")(0),x))
namePair1: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[53] at
map at <console>:25

scala> val firstName = namePair.keys
firstName: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[54] at keys at <c
onsole>:25

scala> val firstName1 = namePair1.keys
firstName1: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[55] at keys at <
console>:25

scala> firstName.intersection(firstName1).collect.foreach(println)
Xi
```

9. Take values from 1 to 5 into a RDD. Multiply all the numbers (i.e. 1×2×3×4×5). Persist the result RDD into disk.

Hint: Use reduce() to multiply then convert the result into RDD then use persist() to store RDD

```
val input= sc.parallelize(List(1, 2, 3, 4, 5))
val result = input.reduce((x, y) => x * y)      //  Multiply all the numbers in RDD
```

```
scala> val input= sc.parallelize(List(1, 2, 3, 4, 5))
input: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[62] at parallelize
at <console>:24

scala> val result = input.reduce((x, y) => x * y)
result: Int = 120
```

10. Write the above application (Question 8) into a Scala class file. Read the input from HDFS. Provide spark submit command to execute the class in a Spark cluster. User should provide two HDFS paths as two parameters in the spark-submit command line.

Code in scala Eclipse:

Task10.scala:

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

object task10 {
   def main(args: Array[String]) {
     val input = args(0)
     val input1 = args(1)
     val conf = new SparkConf().setAppName("reduce")
     // Create a Scala Spark Context.
     val sc = new SparkContext(conf)
     val person = sc.textFile(input)
     val person1 = sc.textFile(input1)
     val namePair = person.map(x => (x.split(" ")(0),x))
     val namePair1 = person1.map(x => (x.split(" ")(0),x))
     val firstName = namePair.keys
     val firstName1 = namePair1.keys
     firstName.intersection(firstName1).collect.foreach(println)
   }
}
```

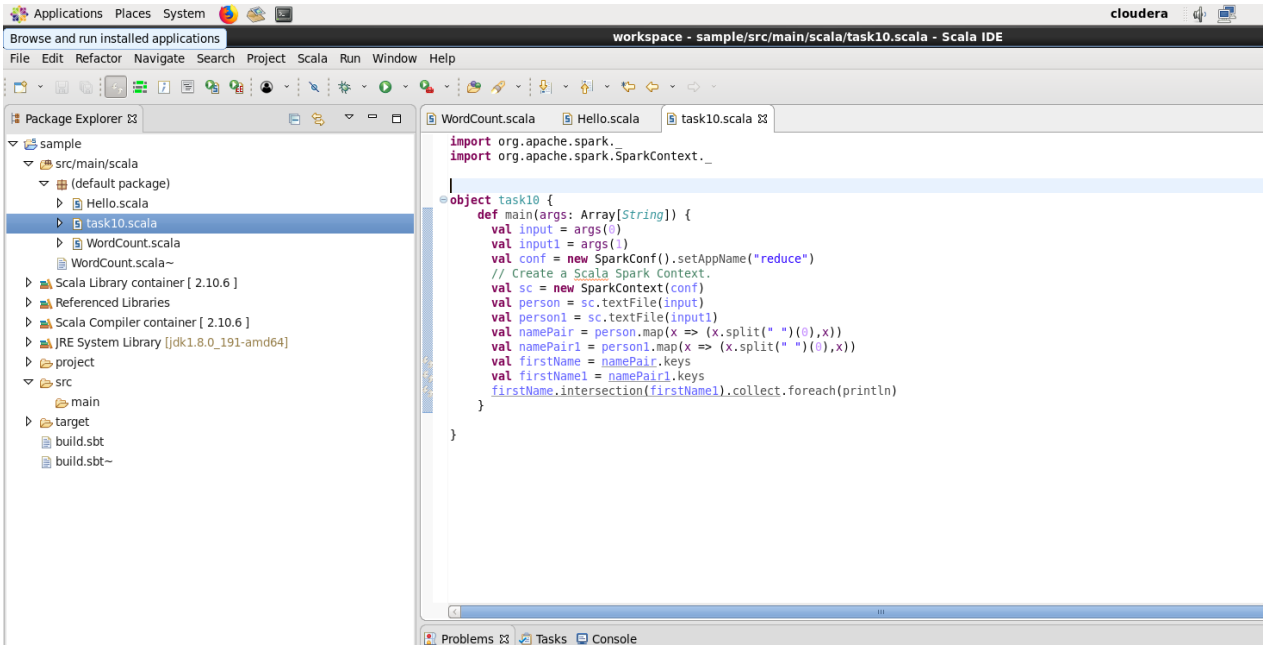
Code in command terminal:

```
$cd /tmp/bigdata/
$sbt clean package
//
$spark-submit --class task10 --master local --deploy-mode client
/tmp/bigdata/target/scala-2.10/sample_2.10-0.1.0-SNAPSHOT.jar /tmp/person.txt /tmp/person1.txt
```

Output:
Xi

ScreenShot:

In scala eclipse:



In spark-shell, prepare the jar:

```
[cloudera@localhost bigdata]$ sbt clean package
[info] Loading settings for project global-plugins from plugins.sbt ...
[info] Loading global plugins from /home/cloudera/.sbt/1.0/plugins
[info] Loading project definition from /tmp/bigdata/project
[info] Loading settings for project bigdata from build.sbt ...
[info] Set current project to sample (in build file:/tmp/bigdata/)
[success] Total time: 0 s, completed Dec 12, 2018 10:29:36 PM
[info] Updating ...
[info] Done updating.
[warn] There may be incompatibilities among your library dependencies.
[warn] Run 'evicted' to see detailed eviction warnings
[info] Compiling 3 Scala sources to /tmp/bigdata/target/scala-2.10/classes ...
[info] Done compiling.
[warn] Multiple main classes detected.  Run 'show discoveredMainClasses' to see
the list
[info] Packaging /tmp/bigdata/target/scala-2.10/sample_2.10-0.1.0-SNAPSHOT.jar .
..
[info] Done packaging.
[success] Total time: 32 s, completed Dec 12, 2018 10:30:08 PM
```

Start calculate:

```
[cloudera@localhost bigdata]$ spark-submit --class task10 --master local --deplo
y-mode client /tmp/bigdata/target/scala-2.10/sample_2.10-0.1.0-SNAPSHOT.jar /tmp
/person.txt /tmp/person1.txt
2018-12-12 23:00:23 WARN  NativeCodeLoader:62 - Unable to load native-hadoop lib
rary for your platform... using builtin-java classes where applicable
2018-12-12 23:00:24 INFO  SparkContext:54 - Running Spark version 2.3.2
2018-12-12 23:00:24 INFO  SparkContext:54 - Submitted application: reduce
2018-12-12 23:00:24 INFO  SecurityManager:54 - Changing view acls to: cloudera
2018-12-12 23:00:24 INFO  SecurityManager:54 - Changing modify acls to: cloudera
2018-12-12 23:00:24 INFO  SecurityManager:54 - Changing view acls groups to:
2018-12-12 23:00:24 INFO  SecurityManager:54 - Changing modify acls groups to:
2018-12-12 23:00:24 INFO  SecurityManager:54 - SecurityManager: authentication d
isabled; ui acls disabled; users  with view permissions: Set(cloudera); groups w
ith view permissions: Set(); users  with modify permissions: Set(cloudera); grou
ps with modify permissions: Set()
```

…

…

```
2018-12-12 23:00:29 INFO  DAGScheduler:54 - Job 0 finished: collect at task10.sc
ala:18, took 1.159355 s
Xi          ←Result
2018-12-12 23:00:29 INFO  SparkContext:54 - Invoking stop() from shutdown hook
2018-12-12 23:00:29 INFO  AbstractConnector:318 - Stopped Spark@24a249d7{HTTP/1.
1,[http/1.1]}{0.0.0.0:4040}
2018-12-12 23:00:29 INFO  SparkUI:54 - Stopped Spark web UI at http://localhost.
localdomain:4040
2018-12-12 23:00:29 INFO  MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMa
sterEndpoint stopped!
2018-12-12 23:00:29 INFO  MemoryStore:54 - MemoryStore cleared
2018-12-12 23:00:29 INFO  BlockManager:54 - BlockManager stopped
2018-12-12 23:00:29 INFO  BlockManagerMaster:54 - BlockManagerMaster stopped
2018-12-12 23:00:29 INFO  OutputCommitCoordinator$OutputCommitCoordinatorEndpoin
t:54 - OutputCommitCoordinator stopped!
2018-12-12 23:00:29 INFO  SparkContext:54 - Successfully stopped SparkContext
2018-12-12 23:00:29 INFO  ShutdownHookManager:54 - Shutdown hook called
2018-12-12 23:00:29 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark
-781830cc-6a08-4720-998b-9010125eb467
2018-12-12 23:00:29 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark
-df4a4fe7-37b0-4ec9-8f8d-cf42381121d4
[cloudera@localhost bigdata]$
```