Hadoop Training Apache Spark

Working with Apache Spark

During this exercises, you become an Apache Spark Developer/Analyst who will create physical objects in HDFS and perform some operations on them.

Getting ready

1. Connect to edge node

Please use your Linux account at the edge node (cdh00.cl.ii.pw.edu.pl)

```
ssh <USERNAME>@cdh00.cl.ii.pw.edu.pl
```

2. Launch spark-shell with the following parameters:

```
spark-shell --master yarn --deploy-mode client --executor-memory 2048m
--num-executors 4 --conf spark.ui.port=95<YOUR-NUMBER>
```

3. You may face an error like this:
   ```
   org.apache.hadoop.security.AccessControlException: Client cannot
   authenticate via:[TOKEN, KERBEROS].
   ```

You will need to generate Kerberos ticket. When prompted provide your password and afterwards verify that ticket has been generated.

```
kinit

Password for <USERNAME>@CL.II.PW.EDU.PL:

klist
```

4. Once again launch spark-shell with the following parameters:

```
spark-shell --master yarn --deploy-mode client --executor-memory 2048m
--num-executors 4 --conf spark.ui.port=95<YOUR-NUMBER>

#change the log level to INFO
scala>sc.setLogLevel("INFO")
```

5. To verify that your spark application is running run in another session command:

```
yarn top
```

6. Read the CSV file into Spark RDD

```
scala> val input =
sc.textFile("/data/datascience/measured_data/measured_data.csv")
```

7. Print the first record from the RDD:

```
scala>input.first

res1: String = 1|2015-05-18
19:24:00|236|0.09920929584628235|kW|s|D|Warsaw-Dereniowa
```

8. Create a new RDD with the record split into separate columns

```
scala> val rdd = input.map(_.split('|') )
scala>rdd.first

res2: Array[String] = Array(1, 2015-05-18 19:24:00, 236,
0.09920929584628235, kW, s, D, Warsaw-Dereniowa)
```

9. Transform the RDD into the new RDD of MDRecord objects:

```
scala> import java.sql.Timestamp

scala> case class
MDRecord(MD_LSB_ID:Int,MD_TIMESTAMP:java.sql.Timestamp,MD_LSB_ID2:Int,
MD_VALUE:Double,MD_UNIT:String, MD_TIMETYPE:String, MD_QUALITY_MARK:String,
MD_DESC:String)

scala> val classRdd=rdd.map(r=>MDRecord(r(0).toInt,
java.sql.Timestamp.valueOf(r(1)),r(2).toInt,r(3).toDouble, r(4), r(5),
r(6), r(7)) )
```

10. Count the number of objects in the RDD (before and after caching):

```
scala> classRdd.count

17/05/19 15:02:46 INFO scheduler.DAGScheduler: Job 0 finished: count at
<console>:33, took 8.404787 s
res1: Long = 7200000

scala> classRdd.cache.count

scala> classRdd.count
INFO scheduler.DAGScheduler: Job 2 finished: count at <console>:33, took
0.102390 s
res3: Long = 7200000
```

11. Check how many records have MD_VALUE greater than 0.5:

```
scala> classRdd.filter(_.MD_VALUE>0.5).count
17/05/19 15:07:51 INFO scheduler.DAGScheduler: Job 4 finished: count at
<console>:33, took 0.228772 s
res5: Long = 3598755
```

12. Calculate average value per year:

```
 scala> val avgPerYear=classRdd.map{case r: MDRecord =>
 (r.MD_TIMESTAMP.getYear+1900,(r.MD_VALUE,1))
 }.reduceByKey((a,b)=>(a._1+b._1,a._2+b._2)).mapValues(r=>r._1/r._2).colle
 ct

 17/05/19 15:08:42 INFO scheduler.DAGScheduler: Job 5 finished: collect at
 <console>:32, took 0.995223 s
 avgPerYear: Array[(Int, Double)] = Array((2016,0.4997122747767127),
 (2017,0.49987841186704357), (2015,0.4999217851067482))
```

13. Remove the cached RDD from memory:

```
scala> classRdd.unpersist()
```

14. Create a RDD from local variable and check the output of map and flatMap transformations:

```
scala> val rdd = sc.parallelize ( (1 to 1000).map(r=>((1 to r).map(_*r)) )
)
scala> rdd.take(3)
```

```
res36: Array[scala.collection.immutable.IndexedSeq[Int]] = Array(Vector(1),
Vector(2, 4), Vector(3, 6, 9))

 scala> val rddFlat = sc.parallelize ( (1 to 1000).flatMap(r=>((1 to
r).map(_*r)) ) )
 scala> rddFlat.take(3)
 res37: Array[Int] = Array(1, 2, 4)
```

15. Join 2 RDD

```
 scala> val input1 = (1000 to 2000).zipWithIndex.map(r=>(r._2,r._1))
 scala> val input2 = (3000 to 4000).zipWithIndex.map(r=>(r._2,r._1))

 scala> val rdd1=sc.parallelize(input1)
 scala> val rdd2=sc.parallelize(input2)
 scala> rdd1.join(rdd2).take(5)

 res42: Array[(Int, (Int, Int))] = Array((384,(1384,3384)),
 (692,(1692,3692)), (356,(1356,3356)), (772, (1772,3772)),
 (324,(1324,3324)))
```