CS6240

Rachna Reddy Melacheruvu

HW 3

https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig

https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins

Citations

1. Joins inspired from book – High Performance Spark by Rachel Warren; Holden Karau
2. Writing output to files - https://stackoverflow.com/questions/44537889/write-store-dataframe-in-text-file

URLs:

Combining in Spark -

1. **groupByKey –**
   Project -
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/groupByKey/Spark-Demo
   Output –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/groupByKey/Spark-Demo/output

2. **reduceByKey-**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/reduceByKey/Spark-Demo
   Ouput –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/reduceByKey/Spark-Demo/output

3. **foldByKey –**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/foldByKey/Spark-Demo
   Output –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/foldByKey/Spark-Demo/output

4. **aggregateByKey –**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/aggregateByKey/Spark-Demo

Output –
https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/aggregateByKey/Spark-Demo/output

5. **groupBy –**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/groupBy/Spark-Demo
   Output –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Combinig/groupBy/Spark-Demo/output

Joins in Spark –

1. **RS – R**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RSRDD/Spark-Demo
   Output –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RSRDD/Spark-Demo/30000-6/output
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RSRDD/Spark-Demo/30000-11/output
   Logs –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RSRDD/Spark-Demo/30000-6/stderr.txt
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RSRDD/Spark-Demo/30000-11/stderr.txt

2. **RS -D**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RSD/Spark-Demo
   Output –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RSD/Spark-Demo/55000-6/output
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RSD/Spark-Demo/55000%20-%2011/output
   Logs –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RSD/Spark-Demo/55000-6/stderr.txt

Large Cluster –
https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RSD/Spark-Demo/55000%20-%2011/stderr.txt

3. **Rep – R**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RepR/Spark-Demo
   Output –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RepR/Spark-Demo/20000-6/output
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RepR/Spark-Demo/20000-11/output
   Logs –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RepR/Spark-Demo/20000-6/stderr.txt
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RepR/Spark-Demo/20000-11/stderr.txt

4. **Rep - D**
   Project –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RepD/Spark-Demo
   Output –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RepD/Spark-Demo/55000-6/output
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment%203/Joins/RepD/Spark-Demo/55000-11/output
   Logs –
   Small Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RepD/Spark-Demo/55000-6/stderr.txt
   Large Cluster –
   https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment%203/Joins/RepD/Spark-Demo/55000-11/stderr.txt

Combining in Spark

1. **groupByKey**

   Pseudo-code

InputFile.map(line => line.split(“,”)(1)) //extract the 2nd element of the split string based on “,”

// for each line in input file.
.map(user => (user,1)) // a pair of RDDs with key as user and value as 1
.groupByKey() // groups the values by key as (k,(v1,v2,v3))
.mapValues(x=>x.sum) // grouped values are summed
Output (user, number_of_followers) to the outputfile.

Usage of toDebugString

```
2019-02-23 20:07:52 INFO  FileInputFormat:249 - Total input paths to process : 1
2019-02-23 20:07:52 INFO  root:30 - (40) MapPartitionsRDD[5] at mapValues at UserFollower.scala:29 []
 |  ShuffledRDD[4] at groupByKey at UserFollower.scala:28 []
 +-(40) MapPartitionsRDD[3] at map at UserFollower.scala:27 []
    |  MapPartitionsRDD[2] at map at UserFollower.scala:26 []
    |  input MapPartitionsRDD[1] at textFile at UserFollower.scala:25 []
    |  input HadoopRDD[0] at textFile at UserFollower.scala:25 []
```

2. **reduceByKey**

Pseudo-code

InputFile.map(line => line.split(“,”)(1)) //extract the 2nd element of the split string based on “,”

// for each line in input file.
.map(user => (user,1)) // a pair of RDDs with key as user and value as 1
.reduce(group by key and sum the values)
Output (user, number_of_followers) to the outputfile.

Usage of toDebugString

```
2019-02-23 20:43:38 INFO  FileInputFormat:249 - Total input paths to process : 1
2019-02-23 20:43:39 INFO  root:29 - (40) ShuffledRDD[4] at reduceByKey at UserFollower.scala:28 []
 +-(40) MapPartitionsRDD[3] at map at UserFollower.scala:27 []
    |  MapPartitionsRDD[2] at map at UserFollower.scala:26 []
    |  input MapPartitionsRDD[1] at textFile at UserFollower.scala:25 []
    |  input HadoopRDD[0] at textFile at UserFollower.scala:25 []
```

3. **foldByKey**

Pseudo-code

InputFile.map(line => line.split(“,”)(1)) //extract the 2nd element of the split string based on “,”

// for each line in input file.
.map(user => (user,1)) // a pair of RDDs with key as user and value as 1
.foldByKey(0)(_+_) // initial values as 0, for each key, sum the values
Output (user, number_of_followers) to the outputfile.

Usage of toDebugString

```
2019-02-23 20:54:34 INFO  FileInputFormat:249 - Total input paths to process : 1
2019-02-23 20:54:36 INFO  root:29 - (40) ShuffledRDD[4] at foldByKey at UserFollower.scala:28 []
```

```
 +-(40) MapPartitionsRDD[3] at map at UserFollower.scala:27 []
    |  MapPartitionsRDD[2] at map at UserFollower.scala:26 []
    |  input MapPartitionsRDD[1] at textFile at UserFollower.scala:25 []
    |  input HadoopRDD[0] at textFile at UserFollower.scala:25 []
```

4. **aggregateByKey**

Pseudo-code

```
InputFile.map(line => line.split(",")(1))    //extract the 2nd element of the split string based on ","
                                             // for each line in input file.
  .map(user => (user,1))                     // a pair of RDDs with key as user and value as 1
  .aggreggateByKey(0)(_+_ , _+_)             // initial values as 0, sum the values before and after shuffle
```

Output (user, number_of_followers) to the outputfile.

Usage of toDebugString

```
2019-02-23 21:10:42 INFO  FileInputFormat:249 - Total input paths to process : 1
2019-02-23 21:10:43 INFO  root:29 - (40) ShuffledRDD[4] at aggregateByKey at UserFollower.scala:28 []
 +-(40) MapPartitionsRDD[3] at map at UserFollower.scala:27 []
    |  MapPartitionsRDD[2] at map at UserFollower.scala:26 []
    |  input MapPartitionsRDD[1] at textFile at UserFollower.scala:25 []
    |  input HadoopRDD[0] at textFile at UserFollower.scala:25 []
```

5. **groupBy**

Pseudo-code

```
InputFile.map(line => line.split(",")(1))    //extract the 2nd element of the split string based on ","
                                             // for each line in input file.
  .map(user => (user,1))                     // a pair of RDDs with key as user and value as 1
  .toDS()                                    // convert pair RDD to DataSet
  .groupBy("_1")                             // group the keys in column with name "_1"
  .count()                                   // count the number of instances
```

Output (user, number_of_followers) to the outputfile.

Usage of explain

```
== Parsed Logical Plan ==
Aggregate [_1#3], [_1#3, count(1) AS count#9L]
+- AnalysisBarrier
      +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType,
fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._1, true, false) AS _1#3,
assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._2 AS _2#4]
         +- ExternalRDD [obj#2]

== Analyzed Logical Plan ==
_1: string, count: bigint
Aggregate [_1#3], [_1#3, count(1) AS count#9L]
+- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString,
assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._1, true, false) AS _1#3,
assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._2 AS _2#4]
   +- ExternalRDD [obj#2]

== Optimized Logical Plan ==
Aggregate [_1#3], [_1#3, count(1) AS count#9L]
+- Project [_1#3]
```

```
   +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString,
assertnotnull(input[0, scala.Tuple2, true])._1, true, false) AS _1#3, assertnotnull(input[0, scala.Tuple2, true])._2
AS _2#4]
      +- ExternalRDD [obj#2]

== Physical Plan ==
*(2) HashAggregate(keys=[_1#3], functions=[count(1)], output=[_1#3, count#9L])
+- Exchange hashpartitioning(_1#3, 200)
   +- *(1) HashAggregate(keys=[_1#3], functions=[partial_count(1)], output=[_1#3, count#14L])
      +- *(1) Project [_1#3]
         +- *(1) SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType,
fromString, assertnotnull(input[0, scala.Tuple2, true])._1, true, false) AS _1#3, assertnotnull(input[0, scala.Tuple2,
true])._2 AS _2#4]
            +- Scan ExternalRDDScan[obj#2]
```

Based on the above information, **reduceByKey, foldByKey** and **aggregateByKey** perform aggregation on each node of the cluster and then shuffle phase takes place.
Whereas, **groupByKey** function first groups all the values for each key from all nodes and then performs aggregation.Similarly, **groupBy** function first groups by the column and then that is passed as input to the aggregation function.

Joins in Spark

1. **RS- R**

   Pseudo Code

```
   val maxId: Int = 30000                                                    // MAX filter
   val textFile = sc.textFile(args(0))
   val edges = textFile.map(line => line.split(","))
.filter(edge => edge(0).toInt < maxId && edge(1).toInt < maxId)  // split each line on ","  and check if
each user ID  within MAX limit
   val leftRDD = edges.map(word => (word(0), word(1)))          // RDD Pair – user1, user2
   val rightRDD = edges.map(word => (word(1), word(0)))         // RDD Pair – user2, user1
   val path2 = rightRDD.join(leftRDD)                           // Join (user2,user1) with (user1,user2)
   val triangles = path2.map(                                   // for each joined key value pair
   {                                                            // ignore key and take the joined tuple
      case(k,(u,v)) =>{(v,u)}}).join(leftRDD)                   // flip tuple and join with original RDD
      .filter({ case(k,(u,v))=> { u == v }                      // if flipped tuple exists in original RDD,
 }).count()                                                     // it means there is a triangle, count it

   val finalCount = triangles/3
```

2. **RS – D**

   Pseudo Code

```
   val maxId: Int = 55000                                                    // MAX filter
   val textFile = sc.textFile(args(0))
   val edges = textFile.map(line => line.split(","))
                .filter(edge => edge(0).toInt < maxId && edge(1).toInt < maxId)     // split each line on ","
and check if each user ID  within MAX limit
   val leftDF = edges.map(word => (word(0), word(1)))        // RDD pair – user1,user2
                .toDF("user1","user2")                       // convert RDD to DF with column names
   val rightDF = edges.map(word => (word(0), word(1)))      // RDD pair – user1,user2
```

```
                    .toDF("user2","user3")                  // convert RDD to DF with column names
   val path2 = leftDF.join(rightDF,"user2")                 // join based on key – user2
   val checkPath = path2.select("user3","user1")            // select only the required columns
                    .toDF("user1","userDest")               // convert it to DF with column names
   val triangles = checkPath.join(leftDF,"user1").filter($"user2"===$"userDest").count() // Join the above
// created DF with the original DF to check if the joining path exists.
   val finalCount = triangles/3
```

3. **Rep - R**

Pseudo Code

```
   val maxId: Int = 30000                                   // MAX filter
   val textFile = sc.textFile(args(0))
   val edges = textFile.map(line => line.split(","))
.filter(edge => edge(0).toInt < maxId && edge(1).toInt < maxId)  // split each line on ","  and check if
each user ID  within MAX limit
val leftRDD = edges.map(word => (word(0), word(1))).groupBy(_._1).mapValues(_.map(_._2))
// RDD Pair – user1, user2 and then convert to adjacency list
val rightRDD = edges.map(word => (word(1), word(0)))        // RDD Pair – user2, user1
val leftRDDB = rightRDD.sparkContext.broadcast(leftRDD.collectAsMap()) // broadcast leftRDD
val path2 = rightRDD.flatMap({                              // iterate through each record in rightRDD
case(k,v1) => leftRDDB.value.get(k).map{                    // foreach value of key in rightRDD join
                                                            // with leftRDDB's values for the same key
                    x=> (x,v1)
}).flatMap(x=>x)
val triangles = path2.flatMap({                             // iterate through each record in path2
case(k,v1) => leftRDDB.value.get(k).map{                    // foreach value of key in rightRDD join
                                                            // with leftRDDB's values for the same key
                    x=> (x,v1)
 }).flatMap(x=>x).filter({
 case(k,v) => k==v}).count()
  val finalCount = triangles/3
```

4. **Rep – D**

Pseudo Code

```
   val maxId: Int = 55000                                   // MAX filter
   val textFile = sc.textFile(args(0))
   val edges = textFile.map(line => line.split(","))
                .filter(edge => edge(0).toInt < maxId && edge(1).toInt < maxId)    // split each line on ","
and check if each user ID  within MAX limit
   val leftDF = edges.map(word => (word(0), word(1)))       // RDD pair – user1,user2
                    .toDF("user1","user2")                  // convert RDD to DF with column names
   val rightDF = edges.map(word => (word(0), word(1)))      // RDD pair – user1,user2
                    .toDF("user2","user3")                  // convert RDD to DF with column names
   val path2 = rightDF.join(broadcast(leftDF),"user2")      // broadcast DF and join based on key – user2
   val checkPath = path2.select("user3","user1")            // select only the required columns
                    .toDF("user1","userDest")               // convert it to DF with column names
   val triangles = checkPath.join(broadcast(leftDF),"user1")
                    .filter($"user2"===$"userDest").count()
```

```
// Join the above created DF with the original broadcast DF to check if the joining path exists.

val finalCount = triangles/3
```

Running Time Measurements

Configuration	***Measurements***	***Small Cluster Result***	***Large Cluster Result***
RS – R	Running Time	32 min 13 sec	21 min 12 sec
MAX = 30000	Triangle Count	3087524	3087524
RS – D	Running Time	10 min 32 sec	6 min 16 sec
MAX = 55000	Triangle Count	15326693	15326693
Rep – R	Running Time	37 min 41 sec	58 min 43 sec
MAX = 20000	Triangle Count	2411611	2411611
Rep – D	Running Time	3 min 14 sec	2 min 38 sec
MAX = 55000	Triangle Count	15326693	15326693