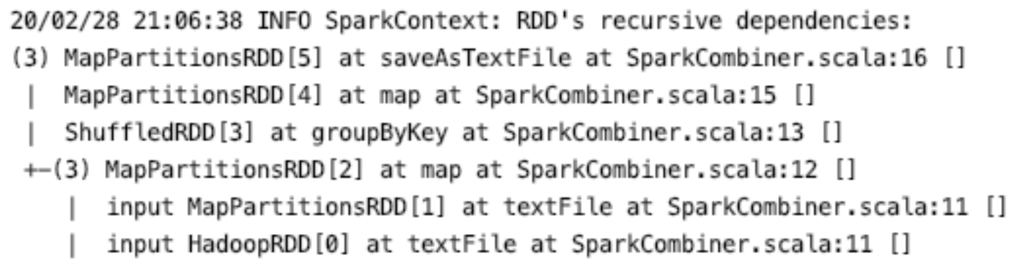
## Combining in Spark:

1. *Pseudo-code Spark Scala - Follower Count:*

```
rddGroupByKey(sc: SparkContext, input: String, output: String):
  inputRDD = sc.textFile(input)
  pairsRDD = inputRDD.map(row => (row.split(“,”)(1), 1))
  groupedRDD = pairsRDD.groupByKey()
  followersCountRDD = groupedRDD.map(x => (x._1, x._2.sum))
  followersCountRDD.saveAsTextFile(output)


rddReduceByKey(sc: SparkContext, input: String, output: String):
   inputRDD = sc.textFile(input)
   pairsRDD = inputRDD.map(row => (row, 1))
   reducedRDD = pairsRDD.reduceByKey(_+_)
   reducedRDD.saveAsTextFile(output)


rddFoldByKey(sc: SparkContext, input: String, output: String):
  inputRDD = sc.textFile(input)
  pairsRDD = inputRDD.map(row => (row.split(“,”)(1), 1))
  foldRDD = pairsRDD.foldByKey(0)(_+_)
  foldRDD.saveAsTextFile(output)


rddAggregateByKey(sc: SparkContext, input: String, output: String):
  inputRDD = sc.textFile(input)
  pairsRDD = inputRDD.map(row => (row.split(“,”)(1), 1))
  aggregateRDD = pairsRDD.aggregateByKey(0)(_+_,_+_)
  aggregateRDD.saveAsTextFile(output)


groupByDataset(spark: SparkSession, input: String, output: String):
  dataSchema = StructType(Array(
    StructField("follower", LongType, nullable = false),
    StructField("user", LongType, nullable = false)))
  inputDF = spark.read.format("csv").schema(dataSchema).load(input)
  groupDF = inputDF.groupBy("user").count()
  groupDF.write.csv(output)
```

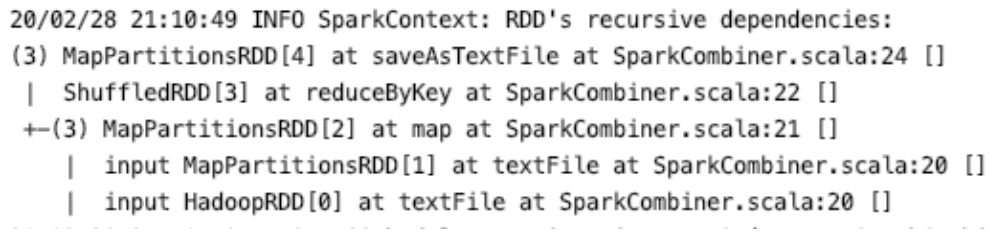
2. *Reporting toDebugString() for RDD-based programs*

- *RDD-G*



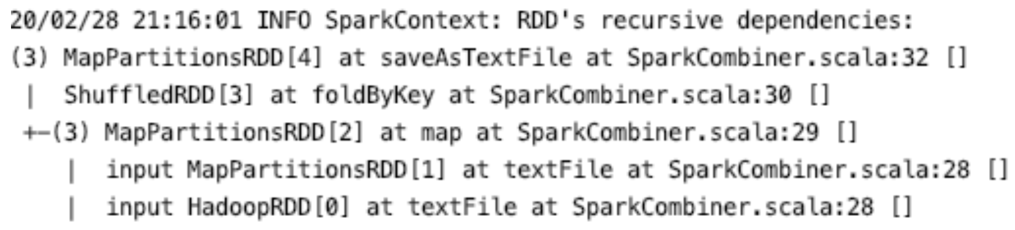
```
20/02/28 21:06:38 INFO SparkContext: RDD's recursive dependencies:
(3) MapPartitionsRDD[5] at saveAsTextFile at SparkCombiner.scala:16 []
 |  MapPartitionsRDD[4] at map at SparkCombiner.scala:15 []
 |  ShuffledRDD[3] at groupByKey at SparkCombiner.scala:13 []
 +-(3) MapPartitionsRDD[2] at map at SparkCombiner.scala:12 []
    |  input MapPartitionsRDD[1] at textFile at SparkCombiner.scala:11 []
    |  input HadoopRDD[0] at textFile at SparkCombiner.scala:11 []
```

- *RDD-R*



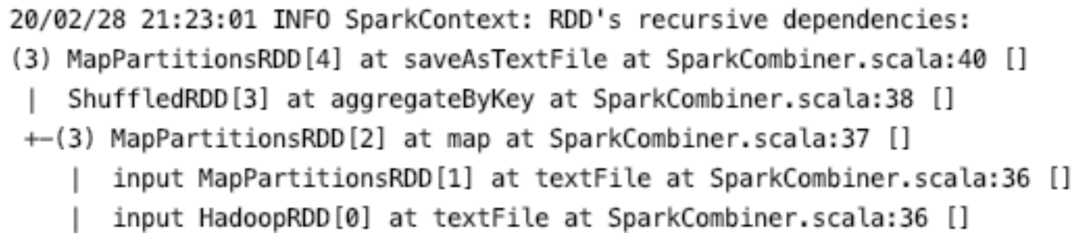
```
20/02/28 21:10:49 INFO SparkContext: RDD's recursive dependencies:
(3) MapPartitionsRDD[4] at saveAsTextFile at SparkCombiner.scala:24 []
 |  ShuffledRDD[3] at reduceByKey at SparkCombiner.scala:22 []
 +-(3) MapPartitionsRDD[2] at map at SparkCombiner.scala:21 []
    |  input MapPartitionsRDD[1] at textFile at SparkCombiner.scala:20 []
    |  input HadoopRDD[0] at textFile at SparkCombiner.scala:20 []
```

- *RDD-F*



```
20/02/28 21:16:01 INFO SparkContext: RDD's recursive dependencies:
(3) MapPartitionsRDD[4] at saveAsTextFile at SparkCombiner.scala:32 []
 |  ShuffledRDD[3] at foldByKey at SparkCombiner.scala:30 []
 +-(3) MapPartitionsRDD[2] at map at SparkCombiner.scala:29 []
    |  input MapPartitionsRDD[1] at textFile at SparkCombiner.scala:28 []
    |  input HadoopRDD[0] at textFile at SparkCombiner.scala:28 []
```

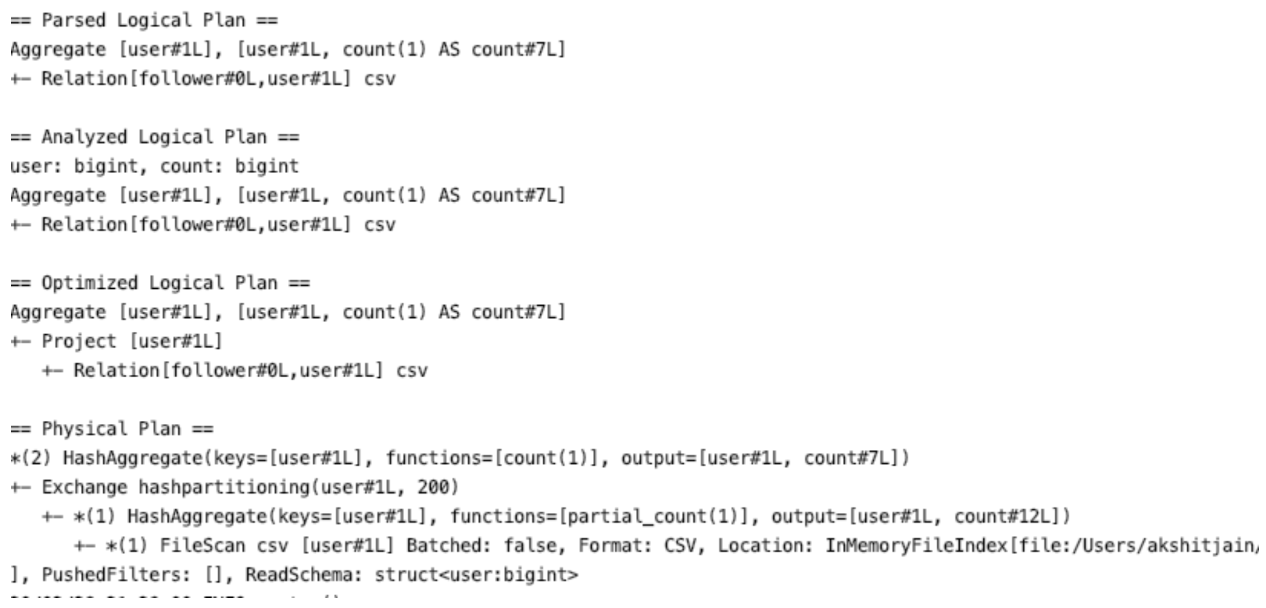
- *RDD-A*



```
20/02/28 21:23:01 INFO SparkContext: RDD's recursive dependencies:
(3) MapPartitionsRDD[4] at saveAsTextFile at SparkCombiner.scala:40 []
 |  ShuffledRDD[3] at aggregateByKey at SparkCombiner.scala:38 []
 +-(3) MapPartitionsRDD[2] at map at SparkCombiner.scala:37 []
    |  input MapPartitionsRDD[1] at textFile at SparkCombiner.scala:36 []
    |  input HadoopRDD[0] at textFile at SparkCombiner.scala:36 []
```

3. *Reporting explain() for Dataset-based programs:*

   - *DSET*



```
== Parsed Logical Plan ==
Aggregate [user#1L], [user#1L, count(1) AS count#7L]
+- Relation[follower#0L,user#1L] csv

== Analyzed Logical Plan ==
user: bigint, count: bigint
Aggregate [user#1L], [user#1L, count(1) AS count#7L]
+- Relation[follower#0L,user#1L] csv

== Optimized Logical Plan ==
Aggregate [user#1L], [user#1L, count(1) AS count#7L]
+- Project [user#1L]
   +- Relation[follower#0L,user#1L] csv

== Physical Plan ==
*(2) HashAggregate(keys=[user#1L], functions=[count(1)], output=[user#1L, count#7L])
+- Exchange hashpartitioning(user#1L, 200)
   +- *(1) HashAggregate(keys=[user#1L], functions=[partial_count(1)], output=[user#1L, count#12L])
      +- *(1) FileScan csv [user#1L] Batched: false, Format: CSV, Location: InMemoryFileIndex[file:/Users/akshitjain,
], PushedFilters: [], ReadSchema: struct<user:bigint>
```

4. *State clearly which of the programs performs aggregation before shuffling, and which does not.*
   - RDD-R, RDD-F and RDD-A perform aggregation before shuffling, whereas RDD-G, and DSET don't.

## Join Implementation:

### *RS-R (Spark Scala Pseudocode):*

```
MAX_FILTER = 50000

rddReduceSideJoin(sc: SparkContext, input: String, output: String):
 rowRDD = sc.textFile(input)
 filterRDD = rowRDD.filter(row => {
  rowVals = row.split(",")
  from = rowVals(0)
  to = rowVals(1)
  from < MAX_FILTER && to < MAX_FILTER
 })
 fromRDD = filterRDD.map(row => {
  rowVals = row.split(",")
  from = rowVals(0)
  to = rowVals(1)
  (from, to)
 })
 toRDD = filterRDD.map(row => {
  rowVals = row.split(",")
  from = rowVals(0)
  to = rowVals(1)
  (to, from)
 })
 counter = Accumulator()
 path2RDD = joinOnKey(fromRDD, toRDD).map(_._2)
 triangleRDD = joinOnKey(path2RDD, fromRDD).map(_._2)
 triangleRDD.foreach { x => if (x._1 == x._2) counter.add(1) }
 print(counter.value / 3)
}

joinOnKey(fromRDD, toRDD):
 joinedRDD = fromRDD.join(toRDD)
 joinedRDD
}
```

*RS-D (Spark Scala Pseudocode):*

```
MAX_FILTER = 50000;

dsReduceSideJoin(spark: SparkSession, input: String, output: String)
  customSchema = StructType(Array(
    StructField("follower_id", LongType, nullable = false),
    StructField("followee_id", LongType, nullable = false)
  ))
 followerDS = spark.read.format("csv").schema(customSchema)
    .load(input)
    .where(follower_id < MAX_FILTER && followee_id < MAX_FILTER)

 path2DS =  followerDS.as("a").joinWith(followerDS.as("b"),  a.followee_id === b.follower_id)

 triangleDS = path2DS.as("a").joinWith(followerDS.as("b"),
      a._1.follower_id === b.followee_id  &&  a._2.followee_id === b.follower_id)
print(triangleDS.count() / 3)
```

*Rep-R (Spark Scala Pseudocode):*

```
MAX_FILTER = 50000;

rddReplicatedJoin(sc: SparkContext, input: String, output: String):
 rowRDD = sc.textFile(input)
 filterRDD = rowRDD.filter(row => {
  rowValues = row.split(",")
  from = rowValues(0)
  to = rowValues(1)
  from.toLong < MAX_FILTER && to.toLong < MAX_FILTER
 })
 fromRDD = filterRDD.map(row => {
  rowValues = row.split(",")
  from = rowValues(0)
  to = rowValues(1)
   (from, to)
  })
 toRDD = filterRDD.map(row => {
  rowValues = row.split(",")
  from = rowValues(0)
  to = rowValues(1)
  (from, to)
 })
 counter = Accumulator()
 edgesRDD = fromRDD.collect().groupBy { case (from, to) => to }
  broadcastVal = sc.broadcast(edgesRDD)
  path2 = toRDD.mapPartitions(iter => {
    iter map {case (from, to) =>
      if (broadcastVal.value.get(from).isDefined) {
        arr  = broadcastVal.value(from)
        arr.foreach { case (mapFrom, mapTo) =>
          if (broadcastVal.value.get(mapFrom).isDefined) {
            arr1 = broadcastVal.value(mapFrom)
            arr1.foreach { case (f, t) =>
              if (to.equals(f)) {
                counter.add(1)
 print(counter.value / 3)
```

*Rep-D (Spark Scala Pseudocode):*

```
MAX_FILTER = 50000;

dsReplicatedJoin(spark: SparkSession, input: String, output: String):
 customSchema = StructType(Array(
  StructField("follower_id", LongType, nullable = false),
  StructField("followee_id", LongType, nullable = false)))

 followersDS = spark.read.format("csv").schema(customSchema).
  load(input)
  .where(follower_id < MAX_FILTER && followee_id < MAX_FILTER)

 path2DS =
  followersDS.as("a").join(broadcast(followersDS).as("b"),
   a.followee_id === b.follower_id).select(a.follower_id, b.followee_id)

 triangleDS =
  path2DS.as("a").joinWith(followersDS.as("b"),
   a.follower_id === b.followee_id && a.followee_id === b.follower_id)

 print(triangleDS.count() / 3)
```

*Comparing Running Time:*

| **Configuration** | **Small Cluster Result** | **Large Cluster Result** |
|---|---|---|
| **RS-R, MAX =** 15000 | Running time: 17 mins<br><br>Triangle count: 1096146 | Running time: 14 mins,<br><br>Triangle count: 1096146 |
| **RS-D, MAX =** 60000 | Running time: 19 mins,<br><br>Triangle count: 2.3801662E7 | Running time: 12 mins,<br><br>Triangle count: 2.3801662E7 |
| **Rep-R, MAX =** 10000 | Running time: 21 mins,<br><br>Triangle count: 520296 | Running time: 18 mins,<br><br>Triangle count: 520296 |
| **Rep-D, MAX =** 180000 | Running time: 10 mins,<br><br>Triangle count: 7.3323616E7 | Running time: 8 mins,<br><br>Triangle count: 7.3323616E7 |

*<u>Links to the Log files:</u>*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/RS-R-Small.txt*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/RS-R-Large.txt*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/RS-D-Small.log*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/RS-D-Large.log*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/Rep-R-Small.log*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/Rep-R-Large.log*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/Rep-D-Small.log*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/logs/Rep-D-Large.log*

*<u>Links to the Output files:</u>*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/RS-R-Small-Output.txt*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/RS-R-Large-Output.txt*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/RS-D-Small-Output.txt*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/RS-D-Large-Output.txt*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/Rep-R-Small-Output.txt*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/Rep-R-Large-Output.txt*

*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/Rep-D-Small-Output.txt*
*/Users/akshitjain/Desktop/Northeastern-University/Large-Scale/spark-twitter-data/output/Rep-D-Large-Output.txt*