# CS6240, MW, Assignment3, Hardik Shah

https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3

## Combining in Spark (Part 1)

**Pseudo Code**

1. RDD-G

```
followerCount(inputFile) {
      sc = new SparkContext(conf)
      textFile = sc.readFile(inputFile)

      /**
       * Reading input line by line,
       * since it is of the form x,y, separated by comma
       * creating a new rdd of form (y,1)
       * since y has x has follower
       */
      followerRDDs = textFile.map(line => {
           userIds = line.split(",")
           followedUserId = userIds(1)
           (followedUserId, 1)
          }

       /**
       * using groupByKey to group all the user
       * and then add the count using reduce
       */
      followerCount = followerRDDs.groupByKey()
                                  .mapValues(noOfFollowers =>
                                        noOfFollowers.reduce((user1,user2)
                                                      => user1 + user2)
                                   )
      followerCount
}
```

2. RDD-R

```
followerCount(inputFile) {
      sc = new SparkContext(conf)
      textFile = sc.readFile(inputFile)

      followerRDDs = textFile.map(line => {
               userIds = line.split(",")
               followedUserId = userIds(1)
               (followedUserId, 1)
            }

      /**
       * using ReduceByKey to group all users and add the counts,
       * performing in-mapper combining
       */
      followerCount = followerRDDS.reduceByKey((user1, user2) =>
                                    user1 + user2)
      followerCount
}
```

3. RDD-F

```
followerCount(inputFile) {
      sc = new SparkContext(conf)
      textFile = sc.readFile(inputFile)

      followerRDDs = textFile.map(line => {
           userIds = line.split(",")
           followedUserId = userIds(1)
           (followedUserId, 1)
      }

      /**
       * using FoldByKey to group all users and add the counts,
       * performing in-mapper combining, with initial count as 0
       */
      followerCount = followerRDDS.foldByKey(0)((user1, user2) =>
                                    user1 + user2)
      followerCount
}
```

4. RDD-A

```
followerCount(inputFile) {
      sc = new SparkContext(conf)
      textFile = sc.readFile(inputFile)

      addCounts = (followerCount1, followerCount2) =>
                  followerCount1 + followerCount2

      addPartitionCount = (partitionCount1, partitionCount2) =>
                  partitionCount1 + partitionCount2

      followerRDDs = textFile.map(line => {
           userIds = line.split(",")
           followedUserId = userIds(1)
           (followedUserId, 1)
      }

      /**
       * using aggregateByKey to group all users and add the counts,
       * performing in-mapper combining using addToCount function,
       * adding partition using sumPartitionCount
       * with initial count as  0
       */
      followerCount = followerRDDS.aggregateByKey(0)
                                  (addCounts, addPartitionCount)
      followerCount
}
```

5. DSET

```
followerCount(inputFile) {
      sc = new SparkContext(conf)
      textFile = sc.readFile(inputFile)

      followerRDDs = textFile.map(line => {
           userIds = line.split(",")
           followedUserId = userIds(1)
           (followedUserId, 1)
      }

      /**
       * Converting rdd to dataset
       */
      sparkSession =  SparkSession.builder().getOrCreate()
      dataset =  sparkSession.createDataset(rdd)


     /**
      * using groupBy to group same users and adding counts
      * using agg
      *
      * groupBy performs aggregation before shuffling
      */
      followerCounts = dataset.groupBy("_1").agg(sum($"_2"))
      followerCounts
}
```

**toDebugString for RDD**

1. RDD-G

```
(40) MapPartitionsRDD[4] at mapValues at RddG.scala:25 []
 |   ShuffledRDD[3] at groupByKey at RddG.scala:25 []
 +-(40) MapPartitionsRDD[2] at map at RddG.scala:20 []
    |   input MapPartitionsRDD[1] at textFile at RddG.scala:18 []
    |   input HadoopRDD[0] at textFile at RddG.scala:18 []
```

2. RDD-R

```
(40) ShuffledRDD[3] at reduceByKey at RDDR.scala:24 []
 +-(40) MapPartitionsRDD[2] at map at RDDR.scala:20 []
    |   input MapPartitionsRDD[1] at textFile at RDDR.scala:19 []
    |   input HadoopRDD[0] at textFile at RDDR.scala:19 []
```

3. RDD-F

```
(40) ShuffledRDD[3] at foldByKey at RDDF.scala:24 []
 +-(40) MapPartitionsRDD[2] at map at RDDF.scala:20 []
    |   input MapPartitionsRDD[1] at textFile at RDDF.scala:19 []
    |   input HadoopRDD[0] at textFile at RDDF.scala:19 []
```

4. RDD-A

```
(40) ShuffledRDD[3] at aggregateByKey at RDDA.scala:28 []
 +-(40) MapPartitionsRDD[2] at map at RDDA.scala:24 []
    |   input MapPartitionsRDD[1] at textFile at RDDA.scala:19 []
    |   input HadoopRDD[0] at textFile at RDDA.scala:19 []
```

**Physical and Logical Plan Dataset**

DSET

```
== Parsed Logical Plan ==
'Aggregate [_1#3], [_1#3, sum('_2) AS sum(_2)#9]
+- 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, sum(_2): bigint
Aggregate [_1#3], [_1#3, sum(cast(_2#4 as bigint)) AS sum(_2)#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, sum(cast(_2#4 as bigint)) AS sum(_2)#9L]
+- 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=[sum(cast(_2#4 as bigint))],
output=[_1#3, sum(_2)#9L])
+- Exchange hashpartitioning(_1#3, 200)
   +- *(1) HashAggregate(keys=[_1#3], functions=[partial_sum(cast(_2#4 as
bigint))], output=[_1#3, sum#17L])
      +- *(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[obj#2]
()
```

**Programs that perform aggregation before shuffling**

1. RDD-R (reduceByKey)
2. RDD-F (foldByKey)
3. RDD-A (aggregateByKey)
4. DSET (groupBy.agg(sum))

**Programs that does not perform aggregation before shuffling**

1. RGG-G (groupByKey)

## Join Implementation (Part 2)

**Pseudo Code**

1. Rs-R

```
RsR(inputFile) {
  MAX = 40000
  sc = new SparkContext(conf)
  textFile = sc.textFile(inputFile)

  XtoY = textFile.map(line => {
            line.split(",")
        })
        .filter(users => users(0).toInt < MAX && users(1).toInt < MAX)
        .map(users => (users(0), users(1)))

 /**
  * creating RDD of form (y,z) by inverting XtoY since
  * we want to join on y to get path of length 2.
  */
  YtoZ = XtoY.map {
    case (user1, user2) => (user2, user1)
  }

 /**
  * Join XtoY and YtoZ on key Y, such that X!=Z
  * and transform the result to ((Z,X),Y) for the next join
  */
  pathLength2 = XtoY.join(YtoZ).filter(joinedRDD => {
    user(X,Z) = joinedRDD._2
    user.X != user.Z
  }).map {
    case (userY, (userZ, userX)) => ((userZ, userX), userY)
  }
```

```
 /**
  * create ZtoX with key as (Z,X) since we want to join on (Z,X)
  */
  ZtoX = XtoY.map {
    case (userZ, userX) => ((userZ, userX), "")
  }

 /**
  * Join X->Y->Z with Z->X
  */
  socialTriangle = pathLength2.join(ZtoX)
  socialTriangle.count()/3
}
```

2. Rs-D

```
RsD(inputFile) {
    MAX = 50000
    sc = new SparkContext(conf)
    sqlContext = new spark.sql.SQLContext(sc)
    textFile = sc.textFile(inputFile)

    XtoY = textFile.map(line => {
                line.split(",")
          })
          .filter(users => users(0).toInt < MAX && users(1).toInt < MAX)
          .map(users => Row(users(0), users(1)))

 /**
  * Creating schema of type (id, val)
  * id: String
  * val: Integer
  * val refers to the followerCount
  */
    schema = new StructType()
      .add(StructField("userIdX", StringType, true))
      .add(StructField("userIdY", StringType, true))
```

```
 /**
  * Converting rdd to dataframe using the schema
  */
    df = sqlContext.createDataFrame(XtoY, schema);

    XtoY = df.select('userIdX as "df1_X", 'userIdY as "df1_Y")
                  .as("XtoY")

    YtoZ = df.select('userIdX as "df2_X", 'userIdY as "df2_Y")
                  .as("YtoZ")

 /**
  * Join XtoY and YtoZ to get length of path 2 on Key Y
  * such that X != Z
  */
    pathLength2 = XtoY.join(YtoZ)
                  .where($"XtoY.df1_Y" === $"YtoZ.df2_X"
                         && $"XtoY.df1_X" =!= $"YtoZ.df2_Y")

 /**
  * Join Path Length 2 and XtoZ on key (Z,X)
  */
    socialTriangle = pathLength2.as("Path")
                     .join(XtoY.as("ZtoX"))
                     .where($"Path.df2_Y" === $"ZtoX.userIdX"
                            && $"Path.df1_X" === $"ZtoX.userIdY")

    socialTriangle.count()/3
}
```

3. Rep-R

```
RepR(inputFile) {
    MAX = 100000
    sc = new SparkContext(conf)
    accum = sc.longAccumulator;
    textFile = sc.textFile(inputFile)

    XtoY = textFile.map(line => {
            line.split(",")
          }).filter(users => users(0).toInt < MAX && users(1).toInt < MAX)
            .map(users => (users(0).toInt, users(1).toInt))

 /**
  * Converting Rdd of form (x,y) to (x, Set(y1,y2...))
  */
    userMap = XtoY.map(rdd => (rdd._1, Set(rdd._2)))
                  .reduceByKey(_ ++ _)

 /**
  * Converting rdd to map and broadcasting it
  */
    broadcastRdd = sc.broadcast(userMap.collect.toMap)

 /**
  * for Rdd (x,y), get all z's that y follows.
  * for each z, get all the x'' that z follows such that z!=x
  * and check if x is present in z's set.
  *
  * Increment counter by 1
  */
   socialTriangleCount = XtoY.map {
     case (userX, userY) => broadcastRdd.value.getOrElse(userY,Set[Int]())
                            .foreach {
                               userZ =>
        if(userZ != userX && broadcastRdd.value.getOrElse(userZ,Set[Int]())
                            .contains(userX)) {
                               accum.add(1)
                            }
        }
   }.collect()
```

```
    accum.value/3
  }
```

4. Rep-D

```
RepD(inputFile) {
    MAX = 100000
    sc = new SparkContext(conf)
    sqlContext = new spark.sql.SQLContext(sc)
    textFile = sc.textFile(inputFile)

    XtoY = textFile.map(line => {
                line.split(",")
          })
          .filter(users => users(0).toInt < MAX && users(1).toInt < MAX)
          .map(users => Row(users(0), users(1)))

 /**
  * Creating schema of type (id, val)
  * id: String
  * val: Integer
  * val refers to the followerCount
  */
    schema = new StructType()
      .add(StructField("userIdX", StringType, true))
      .add(StructField("userIdY", StringType, true))

 /**
  * Converting rdd to dataframe using the schema
  */
    df = sqlContext.createDataFrame(XtoY, schema);

    XtoY = df.select('userIdX as "df1_X", 'userIdY as "df1_Y")
             .as("XtoY")

    YtoZ = df.select('userIdX as "df2_X", 'userIdY as "df2_Y")
             .as("YtoZ")
```

```
 /**
  * Join XtoY and YtoZ to get length of path 2 on Key Y
  * such that X != Z.
  * Using broadcast to tell spark to use broadcast join.
  */
    pathLength2 = XtoY.join(broadcast(YtoZ), $"XtoY.df1_Y" === $"YtoZ.df2_X"
                                          && $"XtoY.df1_X" =!= $"YtoZ.df2_Y")

 /**
  * Join Path Length 2 and XtoZ on key (Z,X)
  * Using broadcast to tell spark to use broadcast join.
  */
    socialTriangle = pathLength2.as("Path")
                     .join(broadcast(XtoY.as("ZtoX")),
                            $"Path.df2_Y" === $"ZtoX.userIdX"
                           && $"Path.df1_X" === $"ZtoX.userIdY")

    socialTriangle.count()/3
}
```

**Results on Small and Large Clusters**

| Configuration | Small Cluster Result | Large Cluster Result |
|---|---|---|
| **Rs-R, Max = 40k** | Running Time = 29 min<br>Count = 4741564<br>Cluster-id: j-3THYZ1WAN95UF | Running Time = 31 min<br>Count = 4741564<br>Cluster-id: j-2DFXFDSBX70SW |
| **Rs-D, Max = 50k** | Running Time = 13 min<br>Count = 12029907<br>Cluster-id: j-24MOFWH2OTUYX | Running Time = 7 min<br>Count = 12029907<br>Cluster-id: j-1YBJK0KLPLJA0 |
| **Rep-R, Max = 100k** | Running Time = 23 min<br>Count = 47594048<br>Cluster-id: j-1JJFMF4KILWK9 | Running Time = 22 min<br>Count = 47594048<br>Cluster-id: j-1631QSJ9RQODY |
| **Rep-D, Max = 100k** | Running Time = 18 min<br>Count = 47594048<br>Cluster-id: j-1TU24NM9K0CZ7 | Running Time = 16 min<br>Count = 47594048<br>Cluster-id: j-TDEEGVHUW8J4 |

**Github Log Links**

1. RS-R Small Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rs-R%2040k%20Small
2. Rs-R Large Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rs-R%2040k%20Large
3. RS-D Small Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rs-D%2050k%20Small
4. Rs-D Large Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rs-D%2050k%20Large
5. Rep-R Small Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rep-R%20100k%20Small
6. Rep-R Large Cluster:https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rep-R%20100k%20Large
7. Rep-D Small Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rep-D%20100k%20Small
8. Rep-D Large Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/tree/master/logs/Rep-D%20100k%20Large

**Github Output Links**

**There is no output generated for all the joins.**

- **The count for number of triangles is printed to driver; using count() in RsR, RsD and RepD.**
- **The count for number of triangles is printed to driver; using global accumulator in RepR.**
- **Hence, output is presented in the stdout log files of the corresponding clusters.**
- **The below links are for stdout files.**

1. RS-R Small Cluster: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rs-R%2040k%20Small/stdout

   `SocialTriangleCount: 4741564`

2. Rs-R Large Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rs-R%2040k%20Large/stdout
   `SocialTriangleCount: 4741564`

3. RS-D Small Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rs-D%2050k%20Small/stdout
   `Social Triangle Count 12029907`

4. Rs-D Large Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rs-D%2050k%20Large/stdout
   `Social Triangle Count 12029907`

5. Rep-R Small Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rep-R%20100k%20Small/stdout
   `SocialTriangle Count 47594048`

6. Rep-R Large Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rep-R%20100k%20Large/stdout
   `SocialTriangle Count 47594048`

7. Rep-D Small Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rep-D%20100k%20Small/stdout
   `Social Triangle Count47594048`

8. Rep-D Large Cluster:
   https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-3/blob/master/logs/Rep-D%20100k%20Large/stdout
   `Social Triangle Count47594048`