# CS6240-Assignment 2

**FALL 2020**

**Rahul Pandey**

**GitHub: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs**

**RDD Follower Count code: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/src/main/scala/twitterFollowerCount**

**RS and Rep Join code:**
**https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/src/main/scala/twitterTriangleCount**

## Combining in Spark (20 points total):

initialize sc as SparkContext with the set of configuration. The sparkContext is use to import the File into the memory. The input row contains data in the for format (userId1, userdId2) where userId1 is the user following user with userId2.
The input row is passed and each row is split into different userId to get the person being followed.

### RDD-G:

1. **pseudo-code :**

After this step we have
(personBeingFollowed,1) being emitted which is further reduced using reduceByKey method
Which groups values by there key and reduce them using the past function i.e sum in this case.

**func(inputFile, outputFile)**

```
textFile = sc.textFile(args(0)) // Read input file
groupCount = textFile
  .map(following => (following.split(",")(1), 1))  // apply split function and map
                                                     key to 1, to account for each
                                                     person following the user2
  .groupByKey()        // Now group the above values by key, so that we have all
                          (key => 1) where key is the person being followed
  .mapValues(value => value.sum) // Map each userdId2 with the sum of all the
                                    values received after group by
groupCount.saveAsTextFile(args(1)) // Save to file to default storage
```

**Implementation:**

```
textFile = sc.textFile(args(0))
groupCount = textFile
  .map(following => (following.split(",")(1), 1))
  .groupByKey()  //groupByKey can cause out of disk problems as data is sent over
the network and collected on the reduce workers.
  .mapValues(value => value.sum)

println("ToDebugString output: " + groupCount.toDebugString)
groupCount.saveAsTextFile(args(1))
```

2. **toDebugString() Output:**

```
2020-10-22 19:16:33 INFO  FileInputFormat:256 - Total input files to process : 1
ToDebugString output: (40) MapPartitionsRDD[4] at mapValues at RDDG.scala:21 []
 |   ShuffledRDD[3] at groupByKey at RDDG.scala:20 []
 +-(40) MapPartitionsRDD[2] at map at RDDG.scala:19 []
    |   input MapPartitionsRDD[1] at textFile at RDDG.scala:17 []
    |   input HadoopRDD[0] at textFile at RDDG.scala:17 []
```

## RDD-R pseudo-code:

1. **pseudo-code :**

```
textFile = sc.textFile(args(0)) // Read input file
followerCount = textFile
  .map(word => (word.split(",")(1), 1)) // apply split function and map key to 1
  .reduceByKey(_ + _)       // Reduce each key value pair (key,1) with key by
                               adding all values per key
```

2. **toDebugString() Output:**

```
2020-10-22 19:29:06 INFO  FileInputFormat:256 - Total input files to process : 1
ToDebugString output: (40) ShuffledRDD[3] at reduceByKey at RDDR.scala:23 []
 +-(40) MapPartitionsRDD[2] at map at RDDR.scala:22 []
    |   input MapPartitionsRDD[1] at textFile at RDDR.scala:20 []
    |   input HadoopRDD[0] at textFile at RDDR.scala:20 []
```

## RDD-F pseudo-code:

1. **pseudo-code :**

```
textFile = sc.textFile(args(0))      // Read input file
followersCount = textFile
  .map(word => (word.split(",")(1), 1))  // apply split function and map key to 1
  .foldByKey(0)(_ + _)                  // Reduce each key value pair (key,1) with
                                           key by adding all values per key
```

2. **toDebugString() Output:**

```
2020-10-22 19:36:03 INFO  FileInputFormat:256 - Total input files to process : 1
ToDebugString output: (40) ShuffledRDD[3] at foldByKey at RDDF.scala:20 []
 +-(40) MapPartitionsRDD[2] at map at RDDF.scala:19 []
    |   input MapPartitionsRDD[1] at textFile at RDDF.scala:17 []
    |   input HadoopRDD[0] at textFile at RDDF.scala:17 []
```

## RDD-A pseudo-code:

1. **pseudo-code :**

```
    textFile = sc.textFile(args(0))             // Read input file
    followersPair = textFile
      .map(following => (following.split(",")(1), 1))  // apply split function and
                                                          map key to 1

    initialCount = 0                           //Starting count of triangle = 0
    addCounts =  (n:Int, v: Int) => n+1       // For each v in input, add 1 to
                                                     previous accumulated value

    //Define aggregate function to add values across partition.
    sumAcrossPartitions =  (n1: Int, n2: Int) => n1+n2

    //Count followers per key, using aggregateByKey, passing the initial count = 0,
   // passing the above defined aggregate function.

    followersCount =  followersPair.aggregateByKey(initialCount)
(addCounts,sumAcrossPartitions)

    println("ToDebugString output: " + followersCount.toDebugString)
    followersCount.saveAsTextFile(args(1))    //Save file to default storage
```

2. **toDebugString() Output:**

```
2020-10-22 19:44:27 INFO  FileInputFormat:256 - Total input files to process : 1
ToDebugString output: (40) ShuffledRDD[3] at aggregateByKey at RDDA.scala:27 []
 +-(40) MapPartitionsRDD[2] at map at RDDA.scala:21 []
    |   input MapPartitionsRDD[1] at textFile at RDDA.scala:19 []
    |   input HadoopRDD[0] at textFile at RDDA.scala:19 []
```

## DSET pseudo-code :

1. **pseudo-code :**

```
//Get the spark session
val spark: SparkSession = SparkSession.builder().master("local").getOrCreate()

//Read file from the input path
dataset: Dataset[String] = spark.read.textFile(args(0))


// Apply split function and map key(userId2) to 1
dataset2 = dataset.map(following => (following.split(",")(1), 1))

//Group the dataset by col _1, i.e userdId2 and count the number of rows, it is
// effectively the number of followers it has.
followersCount = dataset2.groupBy("_1").count()


followersCount.rdd.map(_.toString()).saveAsTextFile(args(1)) //Save the file to
                                                                 default storage
```

2. **explain() Output:**

```
2020-10-23 14:18:47 INFO  FileSourceScanExec:54 - Pushed Filters:
== Parsed Logical Plan ==
Aggregate [_1#9], [_1#9, count(1) AS count#24L]
+- AnalysisBarrier
      +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String,
StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._1, true,
false) AS _1#9, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._2 AS _2#10]
         +- MapElements <function1>, class java.lang.String,
[StructField(value,StringType,true)], obj#8: scala.Tuple2
            +- DeserializeToObject cast(value#0 as string).toString, obj#7: java.lang.String
               +- Project [value#0]
                  +- Relation[value#0] text

== Analyzed Logical Plan ==
_1: string, count: bigint
Aggregate [_1#9], [_1#9, count(1) AS count#24L]
+- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String,
StringType, fromString, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._1, true,
false) AS _1#9, assertnotnull(assertnotnull(input[0, scala.Tuple2, true]))._2 AS _2#10]
   +- MapElements <function1>, class java.lang.String, [StructField(value,StringType,true)],
obj#8: scala.Tuple2
      +- DeserializeToObject cast(value#0 as string).toString, obj#7: java.lang.String
         +- Project [value#0]
            +- Relation[value#0] text

== Optimized Logical Plan ==
Aggregate [_1#9], [_1#9, count(1) AS count#24L]
+- Project [_1#9]
   +- SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String,
StringType, fromString, assertnotnull(input[0, scala.Tuple2, true])._1, true, false) AS
_1#9, assertnotnull(input[0, scala.Tuple2, true])._2 AS _2#10]
      +- MapElements <function1>, class java.lang.String,
[StructField(value,StringType,true)], obj#8: scala.Tuple2
```

```
         +- DeserializeToObject value#0.toString, obj#7: java.lang.String
            +- Relation[value#0] text

== Physical Plan ==
*(2) HashAggregate(keys=[_1#9], functions=[count(1)], output=[_1#9, count#24L])
+- Exchange hashpartitioning(_1#9, 200)
   +- *(1) HashAggregate(keys=[_1#9], functions=[partial_count(1)], output=[_1#9, 
count#29L])
      +- *(1) Project [_1#9]
         +- *(1) SerializeFromObject [staticinvoke(class 
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, assertnotnull(input[0, 
scala.Tuple2, true])._1, true, false) AS _1#9, assertnotnull(input[0, scala.Tuple2, 
true])._2 AS _2#10]
            +- *(1) MapElements <function1>, obj#8: scala.Tuple2
               +- *(1) DeserializeToObject value#0.toString, obj#7: java.lang.String
                  +- *(1) FileScan text [value#0] Batched: false, Format: Text, Location: 
InMemoryFileIndex[file:/home/rahul/Documents/Fall2020/LSPDP/HW3/input], PartitionFilters: 
[], PushedFilters: [], ReadSchema: struct<value:string>
explain() output: ()
```

## **Programs performs aggregation before shuffling**:

- RDD-R (reduceByKey)
- RDD-A (aggregateByKey)
- RDD-F (foldByKey)
- DSET   (groupBy("_1").count())

## **No aggregation before shuffling**:

- RDD-G (groupByKey)

## Join Implementation (48 points total):

### RS-R Join pseudo-code: (RDD)

//To count the number of triangles we do it in steps: First we need to join edges table on itself and complete the Path (X,Y,Z) for all input pairs (X,Y) and (Y,Z). We do this my taking multiple file inputs of edges.csv and using it to generate all possible (X,Y,Z) pairs such that X follows Y and Y follows Z.

```
MAX_FILTER = 40000 // Define MAX_FILTER

textFile1 = sc.textFile(args(0))   //Read edges.csv as texFile1
textFile2 = sc.textFile(args(0))   //Read edges.csv as texFile2

//Apply MAX_FILTER to allow user id only below the defined constant MAX_FILTER.
//Also map each row to make them of form (Y,X), so as to get ready for join on Y

pairRDD1 = textFile1.filter(line => line.split(",")(1).toInt < MAX_FILTER && line.split(",")
(0).toInt < MAX_FILTER)
  .map(line => (line.split(",")(1), line.split(",")(0)))


//Apply MAX_FILTER to allow user id only below the defined constant MAX_FILTER.
//Also map each row to make them of form (X,Y), so as to get ready for join on Y

pairRDD2 = textFile2.filter(line => line.split(",")(1).toInt < MAX_FILTER && line.split(",")
(0).toInt < MAX_FILTER)
  .map(line => (line.split(",")(0), line.split(",")(1)))


//Perform a Join on Y, each row now looks like, (Y, (X,Z))
joinedRDD = pairRDD1.join(pairRDD2) //Creating intermediate path
```

//After Processing the input file now we have intermediate output with Existing XYZ paths, now we need to make sure path ZX also exists so that make the triangle complete. For this we will use multiple inputs. The intermediate file and our original input file.

For this step now we will have two mappers, one mapper will read intermediate file created in step 1 and emit (X,Z) as the key, the other will read edges.csv and emit (X,Z) for each input record (Z,X). This will insure that both the emits with same key reach the reducer and we can be assured that path ZX exists for this pair. For example:

(X,Y), (Y,Z)   Exists.
We want to be sure (Z,X) also exists in edges.csv, so when we read a record from intermediate output we emit (X,Z) as the key stating we have path (X,Y,Z) for this (X,Z) pair.

When we read edges.csv input to this mapper is path (Z,X) to match it with previous mapper key so as to reach same reducer we need to emit (X,Z) I.e
(userId2,userId1)

//First Mapper, to create path XZ

```
mapRDDXZ = joinedRDD.map(joinedRDD=>((joinedRDD._2._1, joinedRDD._2._2),1))
```

//First Mapper, to create path ZX, apply MAX filter

```
mapRDDZX = textFile1.filter(line => line.split(",")(1).toInt < MAX_FILTER && line.split(",")
(0).toInt < MAX_FILTER)
  .map(line => ((line.split(",")(1),line.split(",")(0)),1))



//Counting path 2 count for each (X,Z) pair

countRDD1 = mapRDDXZ.groupByKey().map(line => ((line._1._1, line._1._2), line._2.size))


//Counting path 2 count for each (X,Z) pair
countRDD2 = mapRDDZX.groupByKey().map(line => ((line._1._1, line._1._2), line._2.size))
//Counting path 2 count for each (Z,X) pair
```

// joining tables on common key (X,Z)

```
joinedTriangleCount = countRDD1.join(countRDD2)
```

Now the above values from mapper is joined on common key (X,Z) in the form ((X,Z), (count1, count2)) and we need to count both values separately and our triangle count will be multiplication of both counts as we got all values for which (X,Y,Z) exists.

//Multiplying count for each (X,Z) pair to get total triangle Count

```
totalTriangleCountPerKey = joinedTriangleCount.mapValues(value => value._2*value._1)

// Now add all the multiplied values per key to get the total triangle sum
multiplied = totalTriangleCountPerKey.map(_._2).sum()/3.0

//Save the triangle count to a file
sc.parallelize(Seq(multiplied)).saveAsTextFile(args(1))
```

## <u>RS-D Join pseudo-code</u>: (DataSet)

//To count the number of triangles we do it in steps: First we need to join edges table on itself and complete the Path (X,Y,Z) for all input pairs (X,Y) and (Y,Z). We do this my taking multiple file inputs of edges.csv and using it to generate all possible (X,Y,Z) pairs such that X follows Y and Y follows Z.

```
MAX_FILTER = 40000 // Define MAX_FILTER

spark: SparkSession = SparkSession.builder().master("local").getOrCreate()

//Read edges.csv as texFile1
val dataset: Dataset[String] = spark.read.textFile(args(0))


//Apply MAX_FILTER to allow user id only below the defined constant MAX_FILTER.
//Also map each row to make them of form (Y,X), so as to get ready for join on Y

import spark.implicits._
splitDataset = dataset.filter(line => line.split(",")(1).toInt < MAX_FILTER &&
line.split(",")(0).toInt < MAX_FILTER)
  .map(line => (line.split(",")(0), line.split(",")(1)))


//Apply MAX_FILTER to allow user id only below the defined constant MAX_FILTER.
//Also map each row to make them of form (X,Y), so as to get ready for join on Y

splitDataset2 = dataset.filter(line => line.split(",")(1).toInt < MAX_FILTER &&
line.split(",")(0).toInt < MAX_FILTER)
  .map(line => (line.split(",")(0), line.split(",")(1)))


//Perform a Join on Y, each row now looks like, (Y, (X,Z))
//Creating intermediate path
joinedDataSet = splitDataset.joinWith(splitDataset2, splitDataset.col("_2") ===
splitDataset2.col("_1"))
```

After Processing the input file now we have intermediate output with Existing XYZ paths, now we need to make sure path ZX also exists so that make the triangle complete. For this we will use multiple inputs. The intermediate file and our original input file.

For this step now we will have two mappers, one mapper will read intermediate file created in step 1 and emit (X,Z) as the key, the other will read edges.csv and emit (X,Z) for each input record (Z,X). This will insure that both the emits with same key reach the reducer and we can be assured that path ZX exists for this pair. For example:

(X,Y), (Y,Z)   Exists.
We want to be sure (Z,X) also exists in edges.csv, so when we read a record from intermediate output we emit (X,Z) as the key stating we have path (X,Y,Z) for this (X,Z) pair.

When we read edges.csv input to this mapper is path (Z,X) to match it with previous mapper key so as to reach same reducer we need to emit (X,Z) I.e (userId2,userId1)

//Extract XZ from the dataframe

```
mapRDDXZ = joinedDataSet.map(joinedDS => ((joinedDS._1._1, joinedDS._2._2), 1))
```

//First Mapper, to create path ZX, apply MAX filter

```
mapRDDZX = dataset.filter(line1 => line1.split(",")(1).toInt < MAX_FILTER &&
line1.split(",")(0).toInt < MAX_FILTER)
  .map(line1 => ((line1.split(",")(1), line1.split(",")(0)), 1))


//Counting path 2 count for each (X,Z) pair
countRDD1 = mapRDDXZ.groupByKey(_._1).reduceGroups((x, y) => ((x._1._1, x._1._2), x._2 +
y._2))


//Counting path 2 count for each (X,Z) pair
countRDD2 = mapRDDZX.groupByKey(_._1).reduceGroups((x, y) => ((x._1._1, x._1._2), x._2 +
y._2))
```

// joining tables on common key (X,Z)

```
joinedTriangleCount = countRDD1.joinWith(countRDD2, countRDD1.col("key") ===
countRDD2.col("key"))
```

Now the above values from mapper is joined on common key (X,Z) in the form ((X,Z), (count1, count2)) and we need to count both values separately and our triangle count will be multiplication of both counts as we got all values for which (X,Y,Z) exists.

//Multiplying count for each (X,Z) pair to get triangle count per key

```
triangleCountPerKey = joinedTriangleCount.map(value => value._1._2._2 * value._2._2._2)

// Now add all the multiplied values per key to get the total triangle sum
multiplied = triangleCountPerKey.reduce((x, y) => x + y)/3.0

//Save the triangle count to a file
sc.parallelize(Seq(multiplied)).saveAsTextFile(args(1))
```

## Rep-R pseudo-code for Rep-join (RDD)

The Replicated Join uses the cache to store file and is a map only. For Replicated join we are first going to setup our distributed cache and once we have the edges.csv table in the cache we proceed to mapping part and try to establish path X,Y,Z for each user such that X follows Y, Y follows Z and Z follows X.

The edges.csv is loaded and we try to establish the path such that for each user X we have list of all users X follows. We iterate over the input edges.csv and build this Map. We define it as **broadCastMap**

```
val MAX_FILTER = 80000

//Global counter defined to keep count of triangle Sum

globalTriangleCount = sc.longAccumulator

textFile2 = sc.textFile(args(0))  //Input is the edges.csv file

// Apply MAX_FILTER and map each input row to set of (X,Y)

pairRDD2 = textFile2.filter(line => line.split(",")(1).toInt < MAX_FILTER && line.split(",")
(0).toInt < MAX_FILTER)
  .map(line => (line.split(",")(0), line.split(",")(1)))


// For each user now buld the map such that each user is mapped to a set of user it follows

broadCastMap =   pairRDD2.map(rddRow => (rddRow._1, Set(rddRow._2))).reduceByKey(_++_)


//Broad cast the map value
smallAsMap = sc.broadcast(broadCastMap.collect().toMap)


// For each input user Y, check if the value exists in the map

pairRDD2.foreach(rddRow =>
  if (smallAsMap.value.contains(rddRow._2)) {

    // get list of Z users that Y follows
    zUsersYFollows: Set[String] = smallAsMap.value.get(rddRow._2).get

     zUsersYFollows.foreach(zUser =>
      if (smallAsMap.value.contains(zUser)) { // Get all users Z follows


        //Map each user that z follows as X user
        xUsersZFollow: Set[String] = smallAsMap.value.get(zUser).get

        if (xUsersZFollow.contains(rddRow._1)) { // If Z follows X

               //Increment the triangle count by 1 as we validated the path XYZ
               globalTriangleCount.add(1) //
        }
      })
  })

//Save file to default storage
sc.parallelize(Seq(globalTriangleCount.value/3.0)).saveAsTextFile(args(1))
```

## Rep-D pseudo-code for Rep-join: (Dataset)

The Replicated Join uses the cache to store file and is a map only. For Replicated join we are first going to setup our distributed cache and once we have the edges.csv table in the cache we proceed to mapping part and try to establish path X,Y,Z for each user such that X follows Y, Y follows Z and Z follows X.

The edges.csv is loaded and we try to establish the path such that for each user X we have list of all users X follows. We iterate over the input edges.csv and build this Map. We define it as **broadCastMap**

```
MAX_FILTER = 80000

//Global counter defined to keep count of triangle Sum
globalTriangleCount = sc.longAccumulator

//Input is the edges.csv file
dataset: Dataset[String] = spark.read.textFile(args(0))


// Apply MAX_FILTER and map each input row to set of (X,Y)
splitDataset = dataset.filter(line => line.split(",")(1).toInt < MAX_FILTER &&
line.split(",")(0).toInt < MAX_FILTER)
  .map(line => (line.split(",")(0), line.split(",")(1)))

// For each user now buld the map such that each user is mapped to a set of user it //
follows, _1 and _2 represents follower and followed user

groupByKeyDataset = splitDataset.map(datasetRow => (datasetRow._1,Set(datasetRow._2)))
  .groupByKey(_._1)
  .reduceGroups((x,y) => (x._1,x._2 ++ y._2))
  .map(_._2)

//Broad cast the map value
smallAsMap = sc.broadcast(groupByKeyDataset.collect().toMap)


// For each input user Y, check if the value exists in the map
splitDataset.foreach(dataSetRow =>
  if (smallAsMap.value.contains(dataSetRow._2)) {

 // get list of Z users that Y follows
    zUsersYFollows: Set[String] = smallAsMap.value.get(dataSetRow._2).get

    zUsersYFollows.foreach(zUser =>
      if (smallAsMap.value.contains(zUser)) { // Get all users Z follows

        //Map each user that z follows as X user
        xUsersZFollow: Set[String] = smallAsMap.value.get(zUser).get

        if (xUsersZFollow.contains(dataSetRow._1)) { // If Z follows X

          //Increment the triangle count by 1 as we validated the path XYZ
          globalTriangleCount.add(1)
        }
      }) })

//Save file to default storage
sc.parallelize(Seq(globalTriangleCount.value/3.0)).saveAsTextFile(args(1))
```

| **Configuration** | **Small Cluster Result**<br>**(Configuration: m5.xlarge) 3+1 machines** | **Large Cluster Result**<br>**(Configuration: m5.xlarge) 6+1 machines** |
|---|---|---|
| RS-R, MAX = 40000<br>(Reduced by 10000 as for 50000 It was running forever) | Running time: 17 mins<br>Triangle count: **4741564**<br>(Start: 13:03:24, 13:21:17)<br>j-KDZ2A4SHINYJ | Running time: 10mins<br>Triangle count: **4741564**<br>(Start: 13:38:15, end: 13:48:44)<br>j-KO3MA591TJJ7 |
| RS-D, MAX = 50000 | Running time: 24mins<br>Triangle count: **12029907**<br>(Start: 04:49:01 , End: 05:13:26 ) | Running time: 13 mins<br>Triangle count: **12029907**<br>(Start: 04:14:09, end: 04:27:56 ) |

| | | |
|---|---|---|
| Rep-R, MAX = 80000 | Running time: 23mins<br>Triangle count: **3.5792118E7**<br>**OR 35792118**<br>(Start: 15:14:50, end: 15:37:58 )<br>j-3OY9BBN7YD0UC | Running time: 24mins<br>Triangle count: **3.5792118E7**<br>**OR 35792118**<br>(Start: 16:02:24, end: 16:26:38 )<br>j-NL8O4SG5OJZ0 |
| Rep-D, MAX = 80000 | Running time: 21mins<br>Triangle count: **35792118**<br>(Start: 16:57:07, end: 17:18:59)<br>j-1P17K5YMRDG2Y | Running time: 17mins<br>Triangle count: **35792118**<br>(Start: 17:46:43, end: 18:03:46)<br>j-1ST7FBLNT4OR8 |

**Syslogs Link:** *(Small Cluster: m5.xLarge (3 +1), Large Cluster m5.xLarge(6+1))*

**RDD Logs: (Local run)**

**RDD R**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/RDDLogs/RDD-R.txt

**RDD F**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/RDDLogs/RDD-F.txt

**RDD G**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/RDDLogs/RDD-G.txt

**RDD A** : https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/RDDLogs/RDD-A.txt

**DSET**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/RDDLogs/DSET.txt

***Reduce Side Join:***

**RS-R**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/RSR_Small.txt
Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/RSR_Large.txt

**RS-D**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/RSD_Small.txt

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/RSD_Large.txt

***Replicated Side Join:***

**Rep-R**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/REP-R_Small.txt

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/REP-R_Large.txt

**Rep-D**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/REP-D_Small.txt

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/blob/master/JoinLogs/REP-D_Large.txt

## Output Folder Link: (Including output file) (Local run)

**RDD Logs:**

**RDD R**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRDD-R

**RDD F**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRDD-F

**RDD G**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRDD-G

**RDD A** :https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRDD-A

**DSET**: https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputDSET

***Reduce Side Join:**(Small Cluster: m5.xLarge (3 +1), Large Cluster m5.xLarge(6+1))*

**RS-R**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRSRSmall
Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRSRLarge

**RS-D**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRSDSmall

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRSD2

***Replicated Side Join:** (Small Cluster: m5.xLarge (3 +1), Large Cluster m5.xLarge(6+1))*

**Rep-R**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRepRSmall

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRepRLarge

**Rep-D**
Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRepDSmall

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-3-rahulpandeycs/tree/master/outputRepDLarge