Combing Implementation

//RDD-G, The grouping and aggregation step must be implemented using

groupByKey, followed by the corresponding aggregate function

```
val pairs1 = textFile.flatMap(line => line.split(",")(1)).map(word => (word,
1)).groupByKey().map(t => (t._1,t._2.sum))
pairs1.saveAsTextFile(args(1))
```

```
) MapPartitionsRDD[5] at map at WordCount.scala:42 []
 ShuffledRDD[4] at groupByKey at WordCount.scala:42 []
-(2) MapPartitionsRDD[3] at map at WordCount.scala:42 []
 |  MapPartitionsRDD[2] at flatMap at WordCount.scala:42 []
 |  input MapPartitionsRDD[1] at textFile at WordCount.scala:23 []
 |  input HadoopRDD[0] at textFile at WordCount.scala:23 []
```

data shuffled 2 times

//RDD-R, The grouping and aggregation step must be implemented using reduceByKey.

```
val pairs2 = textFile.flatMap(line => line.split(",")(1)).map(word => (word,
1)).reduceByKey(_ + _)
pairs2.saveAsTextFile(args(1))
```

```
(2) ShuffledRDD[4] at reduceByKey at WordCount.scala:49 []
+-(2) MapPartitionsRDD[3] at map at WordCount.scala:49 []
   |  MapPartitionsRDD[2] at flatMap at WordCount.scala:49 []
   |  input MapPartitionsRDD[1] at textFile at WordCount.scala:23 []
   |  input HadoopRDD[0] at textFile at WordCount.scala:23 []
```

data shuffled 2 times

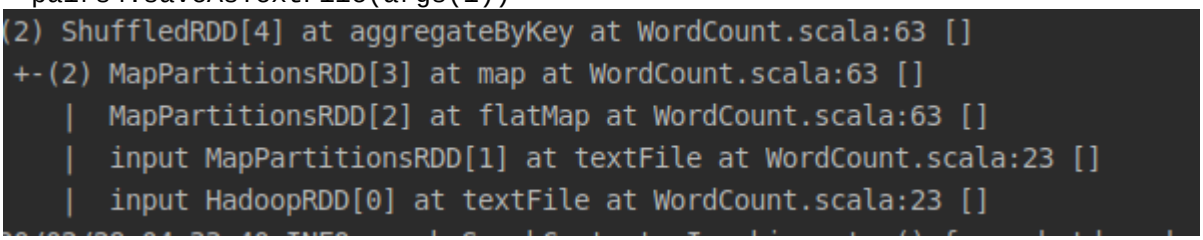
//RDD-F, The grouping and aggregation step must be implemented using foldByKey.

```
val pairs3 = textFile.flatMap(line => line.split(",")(1)).map(word => (word,
1)).foldByKey(0,1)(_ + _)
pairs3.saveAsTextFile(args(1))
```

```
20/02/29 04:31:48 INFO io.SparkHadoopWriter: Job job_20200229043145_0005 committed.
(1) ShuffledRDD[4] at foldByKey at WordCount.scala:56 []
 +-(2) MapPartitionsRDD[3] at map at WordCount.scala:56 []
    |  MapPartitionsRDD[2] at flatMap at WordCount.scala:56 []
    |  input MapPartitionsRDD[1] at textFile at WordCount.scala:23 []
    |  input HadoopRDD[0] at textFile at WordCount.scala:23 []
```

data shuffled 2 times

```
//RDD-A, The grouping and aggregation step must be implemented using
  aggregateByKey.
  val pairs4 = textFile.flatMap(line => line.split(",")(1)).map(word => (word,
  1)).aggregateByKey(0)(_+_,_+_)
  pairs4.saveAsTextFile(args(1))
```



data shuffled 2 times

```
//DSET,  The grouping and aggregation step must be implemented using DataSet,
  with groupBy on the appropriate column, followed by the corresponding
  aggregate function.
  val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()

 import spark.implicits._
 val schema = StructType(Array(StructField("u1", StringType),StructField("u2",
  StringType)))

 val dataset = spark.read
  .format("csv")
  .option("header", false)
  .schema(schema)
  .load(args(0))

  val counts = dataset
  .groupBy($"u2")
  .agg(count($"u2"))
  counts.write.csv(args(1))
```

```
== Parsed Logical Plan ==
'Aggregate ['u2], [unresolvedalias('u2, None), count('u2) AS count(u2)#7]
+- Relation[u1#0,u2#1] csv

== Analyzed Logical Plan ==
u2: string, count(u2): bigint
Aggregate [u2#1], [u2#1, count(u2#1) AS count(u2)#7L]
+- Relation[u1#0,u2#1] csv

== Optimized Logical Plan ==
Aggregate [u2#1], [u2#1, count(u2#1) AS count(u2)#7L]
+- Project [u2#1]
   +- Relation[u1#0,u2#1] csv

== Physical Plan ==
*(2) HashAggregate(keys=[u2#1], functions=[count(u2#1)], output=[u2#1, count(u2)#7L])
+- Exchange hashpartitioning(u2#1, 200)
   +- *(1) HashAggregate(keys=[u2#1], functions=[partial_count(u2#1)], output=[u2#1, count#17L])
```

data shuffled 2 times

Programs with Aggregation before Shuffle:
a) RDD_R
b) RDD_F
c) RDD_A

Programs with Aggregation after Shuffle:
a) RDD_G
b) DSET

Join Implementation

```
def join1(): Unit ={

      //RS-R implements the equivalent of Reduce-side join (hash+shuffle) using RDD
      and pair RDD only

      //max filter
      val max = 10000
      //take the key-value pairs as x->y and y->z

      val input_p1 =
      textFile.map(line=>line.split(",")).map(s=>(s(0),s(1))).filter(t=>
      t._1.toInt<=max & t._2.toInt<=max)
      val input_p2 =
      textFile.map(line=>line.split(",")).map(s=>(s(1),s(0))).filter(t=>
      t._1.toInt<=max & t._2.toInt<=max)
```

```
  //x->y and y->z join
  val first_join = input_p1.join(input_p2)
  //z->x and x->z join
  val input_p3 =
  first_join.map(t=>t.toString()).map(s=>(s.substring(6,7),s.substring(4,5)))
  val second_join = input_p3.join(input_p1)

  val triangle_count = second_join.map(word=>(word,1)).groupByKey().map(t =>
  (t._2.sum)).count()
  sc.parallelize(Seq("Total Number of Triangle",triangle_count.toInt /
  3)).saveAsTextFile(args(1))

}


def join2(): Unit ={

  //RS-D implements the equivalent of Reduce-side join (hash+shuffle) using
  DataSet or DataFrame only

  //max filter
  val max = 100

  //take the key-value pairs as x->y and y->z
  val input_edges =
  textFile.map(line=>line.split(",")).map(s=>(s(0),s(1))).filter(t=>
  t._1.toInt<=max & t._2.toInt<=max).toDS()

  //x->y and y->z join
  val first_join = input_edges.as("table1").join(input_edges.as("table2"))
    .where($"table1._2"===$"table2._1")
    .select($"table1._1",$"table2._2")

  //z->x and x->z join

  val second_join = first_join.as("table3").join(input_edges.as("table1"))
    .where($"table3._1"===$"table1._2" && $"table3._2"===$"table1._1")
    .count()

  sc.parallelize(Seq("No of Triangles", second_join /
  3)).saveAsTextFile(args(1))

}

def join3(): Unit ={

  //Rep-R implements the equivalent of Replicated join (partition+broadcast)
  using RDD and pair RDD only

  // max filter
  val max = 10000

  //take the key-value pairs as x->y and y->z
```

```
      val input_p1 =
      textFile.map(line=>line.split(",")).map(s=>(s(0),s(1))).filter(t=>
      t._1.toInt<=max & t._2.toInt<=max)
      val input_p2 =
      textFile.map(line=>line.split(",")).map(s=>(s(1),s(0))).filter(t=>
      t._1.toInt<=max & t._2.toInt<=max)


      val input_hashmap1 = input_p1.collectAsMap()
      sc.broadcast(input_hashmap1)

      val input_hashmap2 = input_p2.collectAsMap()
      sc.broadcast(input_hashmap2)

      val first_join =

sc.parallelize(input_hashmap1.map(t=>(t._1,t._2)).toSeq).join(sc.parallelize(input_
      hashmap2.map(t=>(t._1,t._2)).toSeq))
      val input_p3 =
      first_join.map(t=>t.toString()).map(s=>(s.substring(6,7),s.substring(4,5)))

      val second_join =
      input_p3.join(sc.parallelize(input_hashmap1.map(t=>(t._1,t._2)).toSeq))
      val triangle_count = second_join.map(word=>(word,1)).groupByKey().map(t =>
      (t._2.sum)).count()
      sc.parallelize(Seq("Total Number of Triangle",triangle_count.toInt /
      3)).saveAsTextFile(args(1))
      sc.parallelize(Seq(second_join.map(word=>(word,1)).groupByKey().map(t =>
      (t._2.sum)).toDebugString)).saveAsTextFile(args(1))

    }


    def join4(): Unit ={

      //RS-D implements the equivalent of Reduce-side join (hash+shuffle) using
      DataSet or DataFrame only
      //max filter
      val max = 10000

      //take the key-value pairs as x->y and y->z

      val input_edges =
      textFile.map(line=>line.split(",")).map(s=>(s(0),s(1))).filter(t=>
      t._1.toInt<=max & t._2.toInt<=max)

      val input_hashmap1 = input_edges.collectAsMap()
      sc.broadcast(input_hashmap1)

      //x->y and y->z join
      val first_join =
sc.parallelize(input_hashmap1.map(t=>(t._1,t._2)).toSeq).toDS().as("table1").join(s
      c.parallelize(input_hashmap1.map(t=>(t._1,t._2)).toSeq).toDS().as("table2"))
        .where($"table1._2"===$"table2._1")
        .select($"table1._1",$"table2._2")
```

```
      //z->x and x->z join

      val second_join =

first_join.as("table3").join(sc.parallelize(input_hashmap1.map(t=>(t._1,t._2)).toSe
      q).toDS().as("table1"))
        .where($"table3._1"===$"table1._2" && $"table3._2"===$"table1._1")
        .count()

      sc.parallelize(Seq("No of Triangles", second_join /
      3)).saveAsTextFile(args(1))

    }
```

Configuration	Small Cluster Result	Large Cluster Result
RS-R, MAX = 20000	Running time: 08:23 – 08:01 (22) Triangle count: 2411611	Running time: 09:26 – 09:12 (14) Triangle count: 2411611
RS-D, MAX = 15000	Running time: 10:59 – 10:55 (5) Triangle count: 1096146	Running time: 09:21 – 09:19 (3) Triangle count: 1096146
Rep-R, MAX = 50000	Running time: Triangle count: 12029907	Running time: 09:22 – 09:02 (18) Triangle count: 12029907
Rep-D, MAX = 35000	Running time: 08:59 – 08:53 (6) Triangle count: 3952871	Running time: Triangle count: 3952871

Deliverable s

main directoy
/home/manjit/Downloads/Spark-Demo/build/deliv/Manjit_Ullal_HW3

drwxr-xr-x 3 manjit manjit 4096 Feb 29 06:27 src
drwxr-xr-x 2 manjit manjit 4096 Feb 29 06:27 output
-rw-r--r-- 1 manjit manjit 2343 Feb 29 06:27 pom.xml
drwxr-xr-x 9 manjit manjit 4096 Feb 29 06:27 logs
-rw-r--r-- 1 manjit manjit 2084 Feb 29 06:27 README.txt
-rw-r--r-- 1 manjit manjit 5221 Feb 29 06:27 Makefile
-rw-r--r-- 1 manjit manjit 173317 Feb 29 06:27 Manjit_ullal_HW3.pdf