CS6240 Homework3

Name: Wenqing Xu

Week1: Combining in Spark

Pseudo-code

RDD-G:

```
val user: RDD[String] = file.flatMap(_.split(",").lastOption)
val counts: RDD[(String,Int)] = user.map(user => (user,1))
val counts2: RDD[(String,Iterable[Int])] = counts.groupByKey()
val results: RDD[(String,Int)] = counts2.map(v => (v._1,v._2.sum))
```

RDD-R:

```
val user: RDD[String] = file.flatMap(_.split(",").lastOption)
val counts: RDD[(String,Int)] = user.map(user => (user,1))
val results: RDD[(String,Int)] = counts.reduceByKey(_+_)
```

RDD-F:

```
val user: RDD[String] = file.flatMap(_.split(",").lastOption)
val counts: RDD[(String,Int)] = user.map(user => (user,1))
val results: RDD[(String,Int)] = counts.foldByKey(0)(_+_)
```

RDD-A:

```
val user: RDD[String] = file.flatMap(_.split(",").lastOption)
val counts: RDD[(String,Int)] = user.map(user => (user,1))
val results: RDD[(String,Int)] = counts.aggregateByKey(0)(_+_,_+_)
```

DSET:

```
val data = sparkSession.read.text(args(0)).as[String]
val user = data.flatMap(r => r.split(",").lastOption)
val counts = user.groupBy("Value")
                 .count().repartition(1)
```

Report and Explaination

RDD-G:

(info: ,(40) MapPartitionsRDD[5] at map at RDD_Group.scala:21 []

| ShuffledRDD[4] at groupByKey at RDD_Group.scala:20 []

+-(40) MapPartitionsRDD[3] at map at RDD_Group.scala:19 []

| MapPartitionsRDD[2] at flatMap at RDD_Group.scala:18 []

| input MapPartitionsRDD[1] at textFile at RDD_Group.scala:17 []

| input HadoopRDD[0] at textFile at RDD_Group.scala:17 [])

RDD-R:

(info: ,(40) ShuffledRDD[4] at reduceByKey at RDD_R.scala:20 []

+-(40) MapPartitionsRDD[3] at map at RDD_R.scala:19 []

| MapPartitionsRDD[2] at flatMap at RDD_R.scala:18 []

| input MapPartitionsRDD[1] at textFile at RDD_R.scala:17 []

| input HadoopRDD[0] at textFile at RDD_R.scala:17 [])

RDD-F:

(info: ,(40) ShuffledRDD[4] at foldByKey at RDD_F.scala:20 []

+-(40) MapPartitionsRDD[3] at map at RDD_F.scala:19 []

| MapPartitionsRDD[2] at flatMap at RDD_F.scala:18 []

| input MapPartitionsRDD[1] at textFile at RDD_F.scala:17 []

| input HadoopRDD[0] at textFile at RDD_F.scala:17 [])

RDD-A:

(info: ,(40) ShuffledRDD[4] at aggregateByKey at RDD_A.scala:20 []

+-(40) MapPartitionsRDD[3] at map at RDD_A.scala:19 []

| MapPartitionsRDD[2] at flatMap at RDD_A.scala:18 []

| input MapPartitionsRDD[1] at textFile at RDD_A.scala:17 []

| input HadoopRDD[0] at textFile at RDD_A.scala:17 [])

DSET:

== Parsed Logical Plan ==

Repartition 1, true

+- Aggregate [Value#7], [Value#7, count(1) AS count#10L]

```
    +- SerializeFromObject [staticinvoke(class
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, input[0,
java.lang.String, true], true, false) AS value#7]
        +- MapPartitions <function1>, obj#6: java.lang.String
            +- DeserializeToObject cast(value#0 as string).toString, obj#5:
java.lang.String
                +- Relation[value#0] text

== Analyzed Logical Plan ==
Value: string, count: bigint
Repartition 1, true
+- Aggregate [Value#7], [Value#7, count(1) AS count#10L]
    +- SerializeFromObject [staticinvoke(class
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, input[0,
java.lang.String, true], true, false) AS value#7]
        +- MapPartitions <function1>, obj#6: java.lang.String
            +- DeserializeToObject cast(value#0 as string).toString, obj#5:
java.lang.String
                +- Relation[value#0] text

== Optimized Logical Plan ==
Repartition 1, true
+- Aggregate [Value#7], [Value#7, count(1) AS count#10L]
    +- SerializeFromObject [staticinvoke(class
org.apache.spark.unsafe.types.UTF8String, StringType, fromString, input[0,
java.lang.String, true], true, false) AS value#7]
        +- MapPartitions <function1>, obj#6: java.lang.String
            +- DeserializeToObject value#0.toString, obj#5: java.lang.String
                +- Relation[value#0] text

== Physical Plan ==
Exchange RoundRobinPartitioning(1)
+- *(3) HashAggregate(keys=[Value#7], functions=[count(1)])
```

+- Exchange hashpartitioning(Value#7, 200)

+- *(2) HashAggregate(keys=[Value#7], functions=[partial_count(1)])

+-*(2)SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString, input[0, java.lang.String, true], true, false) AS value#7]

+- MapPartitions <function1>, obj#6: java.lang.String

+- DeserializeToObject value#0.toString, obj#5: java.lang.String

+- *(1) FileScan text [value#0] Batched: false, Format: Text, Location: InMemoryFileIndex[file:/Users/apple/Desktop/6240/hw3week1/input], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>

RDD method reduceByKey(), foldByKey() and aggregateByKey() perform aggregation before data is shuffled. I actually know this by looking up the source code. They all call the combineByKey() method in their source code. In combineByKey(), it do the aggregation inside. However, groupByKey() does not perform aggregation. As for DSET, we can find in the physical plan that there is a step of aggregation inside each partition.

WEEK2: Join Implementation

pseudo-code:

RS-RDD:

```
filterRDD = filtered (from,to)

fromRDD = filterRDD.map(line => {
    users = line.split(",")
    from = users(0)
    to = users(1)

    (from, to)
})

toRDD = filterRDD.map(line => {
```

```
            users = line.split(",")

            to = users(1)
            from = users(0)

            (to, from)
        })

        answer = Accumulator("Triangle")

        path2 = toRDD.join(fromRDD).map(_._2).filter(line => line._1 != line._2)
        path2.join(toRDD).map(_._2).foreach { x => if (x._1 == x._2) answer.add(1) }
       println("Triangle: " + answer.value / 3)
}

RS-DSET:
       followersDS = filtered (follower_id, user_id) < MAX_FILTER
       path2DS=
            followersDS.as("left").joinWith(followersDS.as("right"),
               " user_id" of left ="follower_id" of right)

      triangleDS=
            path2DS.as("a").joinWith(followersDS.as("b"),
               "follower_id" of a = " user_id" of b and “user_id" of a = “follower_id" of
b)

print("Triangles: " + triangleDS.count() / 3)

REP-RDD:
filterRDD = filtered(from,to) < MAXFILTER

        RDD1 = filterRDD.map(line => {
            val splitVals = line.split(",")
```

```
    from = splitVals(0)
    to = splitVals(1)

    (from, to)
})

RDD2 = filterRDD.map(line => {
    splitVals = line.split(",")

    from= splitVals(0)
    to = splitVals(1)



    (from, to)
})

accum = Accumulator("Triangle Accumulator")

edgesRDD = RDD1.collect().groupBy { case (from, to) => to }
//use the broadcast method to broadcaset the edges
broadCastVal = sc.broadcast(edgesRDD)

path2edges = RDD2.mapPartitions(iter =>
        for each (fromNode,toNode) in RDD2 do
          z = broadCastVal[fromNode]
         // z= all the possible "toNodes" for each fromNode in RDD2
         for each (zfrom,zto) in z do
              x = broadCastVal[zfrom]
         // x= all the possible "toNodes" for each fromNode in z
              for each (f,t) in x do
                   if (f == toNode) do
                        accum += 1
```

println("Triangles:", accum.value / 3)

REP-DSET:

followersDS = filtered (follower_id, user_id) < MAX_FILTER

path2DS: =

followersDS.as("a").join(broadcast(followersDS).as("b"),

"user_id" of a == " follower_id" of b).select($"follower_id" of a, "user_id" of b)

triangleDS =

path2DS.as("a").joinWith(followersDS.as("b"),

" follower_id" of a == "user_id" of b and "user_id" of a== $"follower_id" of b)

print("Triangles: " + triangleDS.count()/3)

Configuration	Small Cluster Result	Large Cluster Result
RS-R, MAX = 20,000	Running time:1318 sec, Triangle count: 2411611	Running time: 918sec, Triangle count: 2411611
RS-D, MAX = 20,000	Running time: 324 sec, Triangle count: 2411611	Running time: 286 sec, Triangle count: 2411611
Rep-R, MAX = 20,000	Running time: 192 sec, Triangle count: 2411611	Running time: 184 sec, Triangle count: 2411611
Rep-R, MAX = 20,000	Running time: 344 sec, Triangle count: 2411611	Running time: 388 sec, Triangle count: 2411611

The time is in the controller file on AWS log folder. The total triangle output is in the console output. (Where to find console output in AWS? Go to containers folder in S3 and click the first folder and select the first container fold, named " container_*****_000001". Open the stdout.gz.)

Week 1 :logs are in the log folder in hw3week1/log

outputs are in hw3week1/output

Week2: each program(RS-R, RS-D, Rep-R, Rep-D) has its own folder, the controller logs are in their own logs folder and output stdout are in their own output folder.