Soumyadeep Sinha
CS6240

Assignment 4

PageRank in Spark (40 points total)

Show the pseudo-code for your PageRank program in Spark Scala. Since many Scala functions are similar to pseudo-code, you may copy-and-paste them here whenever appropriate. (20 points)

RDD:

```
for(i<- 1 to args(2).toInt) {
 ranks.persist()
 links.persist()
 vertex.persist()
 val tempRank = links.join(ranks).map((k) => (k._2._1, k._2._2)).reduceByKey(_ + _)
 val delta: Double = tempRank.filter((x) => (x._1 == "0")).first()._2 / (k * k)
 val temp3 = tempRank.rightOuterJoin(vertex).
   map { case (x, (None, y)) => (x, y)
   case (x, (Some(z), y)) => {
     if (x == "0") {
       ("0", 0.toDouble)
     } else {
       (x, z)
     }
   }
   }
 ranks = temp3.map { (x) => {
   if (x._1 == "0") {
     (x._1, x._2)
   } else {
     (x._1, x._2 + delta)
   }
 }
 }
}
```

DataFrame:

```
for(i<- 1 to args(2).toInt) {
 rankTable.persist()
 linkTable.persist()
 vertexTable.persist()
 val pr = linkTable.join(rankTable, linkTable("src") <=> rankTable("vertex"))
 val temp = pr.drop("vertex").drop("src").groupBy("dest").sum("rank")
 val delta = temp.filter($"dest" === "0").drop("dest").first().getDouble(0) / (k * k)
 val r = vertexTable.join(temp, vertexTable("vertex") === temp("dest"),
"left_outer").drop("dest").drop("notused")
 var nr = r.toDF("v", "r")
 nr = nr.na.fill(0.0)
 nr = nr.select($"v", $"r" + delta).toDF("vertex", "rank")
 nr = nr.withColumn("rank", when(col("vertex") === 0, 0.0).otherwise(col("rank")))
 rankTable = nr
}
```

<u>Report</u>, which of the operations in your program perform an action. (4 points)

1. sum()
2. first()
3. saveAsTextFile(*path*)

Show the lineage for RDD Ranks after 1, 2, and 3 loop iterations. (6 points)

the lineage of 1 iteration

```
(2) MapPartitionsRDD[22] at map at TwitterCount.scala:105 []
 |  MapPartitionsRDD[21] at map at TwitterCount.scala:96 []
 |  MapPartitionsRDD[20] at rightOuterJoin at TwitterCount.scala:95 []
 |  MapPartitionsRDD[19] at rightOuterJoin at TwitterCount.scala:95 []
 |  CoGroupedRDD[18] at rightOuterJoin at TwitterCount.scala:95 []
 |  ShuffledRDD[16] at reduceByKey at TwitterCount.scala:93 []
 +-(2) MapPartitionsRDD[15] at map at TwitterCount.scala:93 []
    |  MapPartitionsRDD[14] at join at TwitterCount.scala:93 []
    |  MapPartitionsRDD[13] at join at TwitterCount.scala:93 []
    |  CoGroupedRDD[12] at join at TwitterCount.scala:93 []
```

```
   +-(2) MapPartitionsRDD[7] at map at TwitterCount.scala:41 []
   |  |      CachedPartitions: 2; MemorySize: 1152.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
   |  |  MapPartitionsRDD[6] at flatMap at TwitterCount.scala:41 []
   |  |  input/graph.txt MapPartitionsRDD[5] at textFile at TwitterCount.scala:41 []
   |  |  input/graph.txt HadoopRDD[4] at textFile at TwitterCount.scala:41 []
   +-(2) MapPartitionsRDD[3] at map at TwitterCount.scala:37 []
      |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
      |  MapPartitionsRDD[2] at flatMap at TwitterCount.scala:37 []
      |  input/rank.txt MapPartitionsRDD[1] at textFile at TwitterCount.scala:37 []
      |  input/rank.txt HadoopRDD[0] at textFile at TwitterCount.scala:37 []
 +-(2) MapPartitionsRDD[11] at map at TwitterCount.scala:45 []
   |  MapPartitionsRDD[10] at flatMap at TwitterCount.scala:45 []
   |  input/vertex.txt MapPartitionsRDD[9] at textFile at TwitterCount.scala:45 []
   |  input/vertex.txt HadoopRDD[8] at textFile at TwitterCount.scala:45 []
```

the lineage of 2 iteration

```
(2) MapPartitionsRDD[33] at map at TwitterCount.scala:105 []
 |  MapPartitionsRDD[32] at map at TwitterCount.scala:96 []
 |  MapPartitionsRDD[31] at rightOuterJoin at TwitterCount.scala:95 []
 |  MapPartitionsRDD[30] at rightOuterJoin at TwitterCount.scala:95 []
 |  CoGroupedRDD[29] at rightOuterJoin at TwitterCount.scala:95 []
 |  ShuffledRDD[27] at reduceByKey at TwitterCount.scala:93 []
 +-(2) MapPartitionsRDD[26] at map at TwitterCount.scala:93 []
   |  MapPartitionsRDD[25] at join at TwitterCount.scala:93 []
   |  MapPartitionsRDD[24] at join at TwitterCount.scala:93 []
   |  CoGroupedRDD[23] at join at TwitterCount.scala:93 []
   +-(2) MapPartitionsRDD[7] at map at TwitterCount.scala:41 []
   |  |      CachedPartitions: 2; MemorySize: 1152.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
   |  |  MapPartitionsRDD[6] at flatMap at TwitterCount.scala:41 []
   |  |  input/graph.txt MapPartitionsRDD[5] at textFile at TwitterCount.scala:41 []
   |  |  input/graph.txt HadoopRDD[4] at textFile at TwitterCount.scala:41 []
   +-(2) MapPartitionsRDD[22] at map at TwitterCount.scala:105 []
      |      CachedPartitions: 2; MemorySize: 1040.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
      |  MapPartitionsRDD[21] at map at TwitterCount.scala:96 []
      |  MapPartitionsRDD[20] at rightOuterJoin at TwitterCount.scala:95 []
      |  MapPartitionsRDD[19] at rightOuterJoin at TwitterCount.scala:95 []
      |  CoGroupedRDD[18] at rightOuterJoin at TwitterCount.scala:95 []
      |  ShuffledRDD[16] at reduceByKey at TwitterCount.scala:93 []
      +-(2) MapPartitionsRDD[15] at map at TwitterCount.scala:93 []
```

```
        |  MapPartitionsRDD[14] at join at TwitterCount.scala:93 []
        |  MapPartitionsRDD[13] at join at TwitterCount.scala:93 []
        |  CoGroupedRDD[12] at join at TwitterCount.scala:93 []
        +-(2) MapPartitionsRDD[7] at map at TwitterCount.scala:41 []
        |  |      CachedPartitions: 2; MemorySize: 1152.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
        |  |  MapPartitionsRDD[6] at flatMap at TwitterCount.scala:41 []
        |  |  input/graph.txt MapPartitionsRDD[5] at textFile at TwitterCount.scala:41 []
        |  |  input/graph.txt HadoopRDD[4] at textFile at TwitterCount.scala:41 []
        +-(2) MapPartitionsRDD[3] at map at TwitterCount.scala:37 []
           |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
           |  MapPartitionsRDD[2] at flatMap at TwitterCount.scala:37 []
           |  input/rank.txt MapPartitionsRDD[1] at textFile at TwitterCount.scala:37 []
           |  input/rank.txt HadoopRDD[0] at textFile at TwitterCount.scala:37 []
     +-(2) MapPartitionsRDD[11] at map at TwitterCount.scala:45 []
        |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
        |  MapPartitionsRDD[10] at flatMap at TwitterCount.scala:45 []
        |  input/vertex.txt MapPartitionsRDD[9] at textFile at TwitterCount.scala:45 []
        |  input/vertex.txt HadoopRDD[8] at textFile at TwitterCount.scala:45 []
 +-(2) MapPartitionsRDD[11] at map at TwitterCount.scala:45 []
    |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
    |  MapPartitionsRDD[10] at flatMap at TwitterCount.scala:45 []
    |  input/vertex.txt MapPartitionsRDD[9] at textFile at TwitterCount.scala:45 []
    |  input/vertex.txt HadoopRDD[8] at textFile at TwitterCount.scala:45 []
```

the lineage of 3 iteration

```
(2) MapPartitionsRDD[44] at map at TwitterCount.scala:105 []
 |  MapPartitionsRDD[43] at map at TwitterCount.scala:96 []
 |  MapPartitionsRDD[42] at rightOuterJoin at TwitterCount.scala:95 []
 |  MapPartitionsRDD[41] at rightOuterJoin at TwitterCount.scala:95 []
 |  CoGroupedRDD[40] at rightOuterJoin at TwitterCount.scala:95 []
 |  ShuffledRDD[38] at reduceByKey at TwitterCount.scala:93 []
 +-(2) MapPartitionsRDD[37] at map at TwitterCount.scala:93 []
    |  MapPartitionsRDD[36] at join at TwitterCount.scala:93 []
    |  MapPartitionsRDD[35] at join at TwitterCount.scala:93 []
    |  CoGroupedRDD[34] at join at TwitterCount.scala:93 []
    +-(2) MapPartitionsRDD[7] at map at TwitterCount.scala:41 []
    |  |      CachedPartitions: 2; MemorySize: 1152.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
    |  |  MapPartitionsRDD[6] at flatMap at TwitterCount.scala:41 []
    |  |  input/graph.txt MapPartitionsRDD[5] at textFile at TwitterCount.scala:41 []
    |  |  input/graph.txt HadoopRDD[4] at textFile at TwitterCount.scala:41 []
```

```
+-(2) MapPartitionsRDD[33] at map at TwitterCount.scala:105 []
  |      CachedPartitions: 2; MemorySize: 1040.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
  |  MapPartitionsRDD[32] at map at TwitterCount.scala:96 []
  |  MapPartitionsRDD[31] at rightOuterJoin at TwitterCount.scala:95 []
  |  MapPartitionsRDD[30] at rightOuterJoin at TwitterCount.scala:95 []
  |  CoGroupedRDD[29] at rightOuterJoin at TwitterCount.scala:95 []
  |  ShuffledRDD[27] at reduceByKey at TwitterCount.scala:93 []
  +-(2) MapPartitionsRDD[26] at map at TwitterCount.scala:93 []
    |  MapPartitionsRDD[25] at join at TwitterCount.scala:93 []
    |  MapPartitionsRDD[24] at join at TwitterCount.scala:93 []
    |  CoGroupedRDD[23] at join at TwitterCount.scala:93 []
    +-(2) MapPartitionsRDD[7] at map at TwitterCount.scala:41 []
    |  |      CachedPartitions: 2; MemorySize: 1152.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
    |  |  MapPartitionsRDD[6] at flatMap at TwitterCount.scala:41 []
    |  |  input/graph.txt MapPartitionsRDD[5] at textFile at TwitterCount.scala:41 []
    |  |  input/graph.txt HadoopRDD[4] at textFile at TwitterCount.scala:41 []
    +-(2) MapPartitionsRDD[22] at map at TwitterCount.scala:105 []
      |      CachedPartitions: 2; MemorySize: 1040.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
      |  MapPartitionsRDD[21] at map at TwitterCount.scala:96 []
      |  MapPartitionsRDD[20] at rightOuterJoin at TwitterCount.scala:95 []
      |  MapPartitionsRDD[19] at rightOuterJoin at TwitterCount.scala:95 []
      |  CoGroupedRDD[18] at rightOuterJoin at TwitterCount.scala:95 []
      |  ShuffledRDD[16] at reduceByKey at TwitterCount.scala:93 []
      +-(2) MapPartitionsRDD[15] at map at TwitterCount.scala:93 []
        |  MapPartitionsRDD[14] at join at TwitterCount.scala:93 []
        |  MapPartitionsRDD[13] at join at TwitterCount.scala:93 []
        |  CoGroupedRDD[12] at join at TwitterCount.scala:93 []
        +-(2) MapPartitionsRDD[7] at map at TwitterCount.scala:41 []
        |  |      CachedPartitions: 2; MemorySize: 1152.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
        |  |  MapPartitionsRDD[6] at flatMap at TwitterCount.scala:41 []
        |  |  input/graph.txt MapPartitionsRDD[5] at textFile at TwitterCount.scala:41 []
        |  |  input/graph.txt HadoopRDD[4] at textFile at TwitterCount.scala:41 []
        +-(2) MapPartitionsRDD[3] at map at TwitterCount.scala:37 []
          |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
          |  MapPartitionsRDD[2] at flatMap at TwitterCount.scala:37 []
          |  input/rank.txt MapPartitionsRDD[1] at textFile at TwitterCount.scala:37 []
          |  input/rank.txt HadoopRDD[0] at textFile at TwitterCount.scala:37 []
      +-(2) MapPartitionsRDD[11] at map at TwitterCount.scala:45 []
        |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
        |  MapPartitionsRDD[10] at flatMap at TwitterCount.scala:45 []
```

```
           |  input/vertex.txt MapPartitionsRDD[9] at textFile at TwitterCount.scala:45 []
           |  input/vertex.txt HadoopRDD[8] at textFile at TwitterCount.scala:45 []
      +-(2) MapPartitionsRDD[11] at map at TwitterCount.scala:45 []
        |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
        |  MapPartitionsRDD[10] at flatMap at TwitterCount.scala:45 []
        |  input/vertex.txt MapPartitionsRDD[9] at textFile at TwitterCount.scala:45 []
        |  input/vertex.txt HadoopRDD[8] at textFile at TwitterCount.scala:45 []
 +-(2) MapPartitionsRDD[11] at map at TwitterCount.scala:45 []
   |      CachedPartitions: 2; MemorySize: 1032.0 B; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
   |  MapPartitionsRDD[10] at flatMap at TwitterCount.scala:45 []
   |  input/vertex.txt MapPartitionsRDD[9] at textFile at TwitterCount.scala:45 []
   |  input/vertex.txt HadoopRDD[8] at textFile at TwitterCount.scala:45 []
```

Explanation:

When we create a new RDD from an existing RDD, the new RDD points to the parent RDD. This is what is called a lineage graph. For example:

If we take the example code from the program:

```
ranks = temp3.map { (x) => {
 if (x._1 == "0") {
   (x._1, x._2)
 } else {
   (x._1, x._2 + delta)
 }
}
```

Here a new rdd is being formed when we make the temp3. As you can see from the output of the debugString(), RDD result keeps a reference to its parent RDD temp3. That is a sort of an RDD lineage.

Discuss how you determined what was actually executed by a job triggered by your program. (2 points)

As we can see from the output of the above debugString, it prints the whole execution plan. Which means that if we read from the last line and go up, we can see which job were triggered by my previous calls.

Report your observations:

(1) Is Spark smart enough to figure out that it can re-use RDDs computed for an earlier action?

No, spark is not smart enough to reuse the RDD.

(2) How do persist() and cache() change this behavior? (4 points)

Persist and cache are optimization techniques which saves the result of RDD evaluation. Using these functions we can save the intermediate result and reduce the overhead.

PageRank in MapReduce (20 points total)

Show the MapReduce pseudo-code. Do not just copy-and paste Java code! (10 points)

Assumption: the value of alpha = 0.

Explanation:

Input: the generated graph, initial ranks, vertices.

We have the graph that i generated, the initial ranking i.e 1/number of vertices and the vertices as input.

(Join graph with the rank)

Mapper1:

Since the rank table is much smaller than the graph table, I made the rank table as a distributed cache and thus used a replicated join to join the graph and the rank table.

Now, we can emit the nodes and node -1's page rank to the reducer.

Reducer1:

In the reducer we try to get the sum of all the page ranks for each node and emit the node and the sum of the page ranks. This is then fed to the next mapper as an input.

Mapper2:

The Output of the previous reducer then becomes the input of this mapper. Here, we join the vertex and the output of the reducer1. This is the place where we also calculate the delta, that is the page rank collected by the dummy node and redistributed among all the non dummy vertices. This mapper emits the node and the new page rank

Reducer 2:

This reducer is used to reset the page rank of the dummy node and emit the final page rank of all the nodes.

This results in a single iteration of the page rank program. To have more iteration, the output of Reducer2 will just become the input of the Mapper1 and the cycle will continue till the desired iteration.

Pseudocode:

```
Class ReplicatedJoinMapper
HashMap rank //Distributed cache //contains the rank
    method Mapper1(doc graph)
        Rank = rank.get(graph.split(“,”)(0));
        emit( graph.split(“,”)(1), Rank)

Class ReplicatedJoinReducer
    Method Reducer1(node, pr[ 0.282, 0.2782…., 02389]):
        Sum = 0.0
        For all count p ∈ pr[ 0.282, 0.2782…., 02389] do :
            sum ← sum + p
        emit(node, sum)

Class ReplicatedJoinMapper2
HashMap ranks //Distributed cache //contains the ranks
    method Mapper2(node,sum)
        delta = ranks.get(“0”)/ (k*k);
        emit( vertex.split(“,”)(0), ranks.get(vertex.split(“,”)(0)+delta)

Class ReplicatedJoinReducer2
    Method Reducer1(node, pr[ 0.282, 0.2782…., 02389]):
        if(node == 0)
            emit(0, 0)
        Else
            For all count p ∈ pr[ 0.282, 0.2782…., 02389] do :
                emit(node,  p)
```

Describe briefly (1 paragraph) how you solved the dangling-page problem. (4 points)

To handle the dangling page problem, I used the dummy page solution where the dummy page will soak up all the contributions and then redistribute it. The Map function emits (dummy, dangling PageRank). The Reduce call for the dummy node then sum up all the page ranks and then pass it as the input to the next job.

Report the running time for each cluster. (2 numbers, 1 point each)

I was not able to run for 9 machines as aws was not able to allocate that many machines to me. So I went for 7 machines instead.

5 cheap machines: 538 sec.

7 cheap machines: 536 sec.

References:
https://data-flair.training/blogs/apache-spark-rdd-persistence-caching/
https://data-flair.training/blogs/rdd-lineage/
https://data-flair.training/blogs/spark-stage/
https://spark.apache.org/docs/latest/rdd-programming-guide.html#actions
https://stackoverflow.com/questions/41763227/remove-a-column-from-a-dataframe-spark
http://www.ccs.neu.edu/home/mirek/classes/CS6240-stable/GraphAlgorithms.pdf