Page Rank in Spark

Algorithms

```
//create the custom graph for given nodes = k
for ( I from 1 to k*k)
{
        if(i%k == 0)
        {
                //mark the dangling node with 0 page rank
                list = list + (I,0)
        }
        else
        {
                list = list + (i+1)
        }
        rank = rank + (I,1/(k*k))
}

//parallelize the list and rank
//run the page rank updation for 10 times

for ( j from 1 to 10)
{
        // join the adjacency list with the rank RDD
        interim = list.join(rank)
        temp =interim.flatmap(
                check the row
                if( row is dangling)
                {
                        list (row id , 0)
                }
                else
                {
                        list(row id)
                }
        //sum the page rank values from all neighbors
        del = temp.reduceby key
        delta = del.lookup(0)(0)

        //update the ranks
        rank = del.map(
                if (row equals 0)
                {
                        ( id,rank)
```

}
else
{
(id, rank + delta/totalnodes)
}

code

```
//create a dummy node to handle the dangling nodes
val dummynode = 0
val initialrank = 1.0 / totalnodes
pgrank = pgrank :+ ((dummynode, 0.0)) //assign dummynode as rank zero

"""create the standard graph
for sample value of k=3, the graph will be, here the 3rd node is the dangling
node
1->2, 2->3
4->5, 5->6
7-> 8 8->9
"""

for (each <- 1 to totalnodes) {
 //assign the kth node as dummy node, therfore divide by k and then assign the
rank as 0
 if (each % numberofnodes == 0) {
 adjacencyList = adjacencyList :+ ((each, dummynode))
 }
 else {
 adjacencyList = adjacencyList :+ ((each, each + 1))
 }
 pgrank = pgrank :+ ((each, initialrank))
}

val iters = 10

val graphrdd = sc.parallelize(adjacencyList)
var rankrdd = sc.parallelize(pgrank)

for (j <- 1 to 10) {
 val contrib = graphrdd.join(rankrdd)
 val interim = contrib.flatMap(row =>
 if (row._1.toInt % numberofnodes == 1) {
 List((row._1, 0.0), row._2)
 } else {
 List(row._2)
 })
 val del = interim.reduceByKey(_ + _)
 //temp2.foreach(println)
 val delta = del.lookup(0)(0)
 rankrdd = del.map(ele =>
 if (ele._1.equals("0")) {
 (ele._1, ele._2)
```

```
    } else {
      (ele._1, ele._2 + (delta / totalnodes))
    }
  )
```

Report, which of the operations in your program perform an action

lookup operation does action

Run the program for 10 iterations for k=100 and report the final PageRanks of the vertices with ID numbers 0 (dummy), 1,…, 99, 100

```
(0,0.010936852726843612)
(1,1.0936852726843613E-6)
(2,2.1765419783124415E-6)
(3,3.248677330419451E-6)
(4,4.310197481020451E-6)
(5,5.3612075311204505E-6)
(6,6.401811541120451E-6)
(7,7.432112541120452E-6)
(8,8.452112541120452E-6)
(9,9.452112541120453E-6)
(10,1.0452112541120453E-5)
(11,1.1045211254112045E-4)
(12,1.1045211254112045E-4)
(13,1.1045211254112045E-4)
(14,1.1045211254112045E-4)
(15,1.1045211254112045E-4)
(16,1.1045211254112045E-4)
(17,1.1045211254112045E-4)
(18,1.1045211254112045E-4)
(19,1.1045211254112045E-4)
(20,1.1045211254112045E-4)
(21,1.1045211254112045E-4)
(22,1.1045211254112045E-4)
(23,1.1045211254112045E-4)
(24,1.1045211254112045E-4)
(25,1.1045211254112045E-4)
(26,1.1045211254112045E-4)
(27,1.1045211254112045E-4)
(28,1.1045211254112045E-4)
(29,1.1045211254112045E-4)
(30,1.1045211254112045E-4)
(31,1.1045211254112045E-4)
(32,1.1045211254112045E-4)
(33,1.1045211254112045E-4)
(34,1.1045211254112045E-4)
(35,1.1045211254112045E-4)
```

(36,1.1045211254112045E-4)
(37,1.1045211254112045E-4)
(38,1.1045211254112045E-4)
(39,1.1045211254112045E-4)
(40,1.1045211254112045E-4)
(41,1.1045211254112045E-4)
(42,1.1045211254112045E-4)
(43,1.1045211254112045E-4)
(44,1.1045211254112045E-4)
(45,1.1045211254112045E-4)
(45,1.1045211254112045E-4)
(47,1.1045211254112045E-4)
(48,1.1045211254112045E-4)
(49,1.1045211254112045E-4)
(50,1.1045211254112045E-4)
(51,1.1045211254112045E-4)
(52,1.1045211254112045E-4)
(53,1.1045211254112045E-4)
(54,1.1045211254112045E-4)
(55,1.1045211254112045E-4)
(56,1.1045211254112045E-4)
(57,1.1045211254112045E-4)
(58,1.1045211254112045E-4)
(59,1.1045211254112045E-4)
(60,1.1045211254112045E-4)
(61,1.1045211254112045E-4)
(62,1.1045211254112045E-4)
(63,1.1045211254112045E-4)
(64,1.1045211254112045E-4)
(65,1.1045211254112045E-4)
(66,1.1045211254112045E-4)
(67,1.1045211254112045E-4)
(68,1.1045211254112045E-4)
(69,1.1045211254112045E-4)
(70,1.1045211254112045E-4)
(71,1.1045211254112045E-4)
(72,1.1045211254112045E-4)
(73,1.1045211254112045E-4)
(74,1.1045211254112045E-4)
(75,1.1045211254112045E-4)
(76,1.1045211254112045E-4)
(77,1.1045211254112045E-4)
(78,1.1045211254112045E-4)
(79,1.1045211254112045E-4)
(80,1.1045211254112045E-4)
(81,1.1045211254112045E-4)
(82,1.1045211254112045E-4)
(83,1.1045211254112045E-4)
(84,1.1045211254112045E-4)
(85,1.1045211254112045E-4)
(86,1.1045211254112045E-4)
(87,1.1045211254112045E-4)

```
(88,1.1045211254112045E-4)
(89,1.1045211254112045E-4)
(90,1.1045211254112045E-4)
(91,1.1045211254112045E-4)
(92,1.1045211254112045E-4)
(93,1.1045211254112045E-4)
(94,1.1045211254112045E-4)
(95,1.1045211254112045E-4)
(96,1.1045211254112045E-4)
(97,1.1045211254112045E-4)
(98,1.1045211254112045E-4)
(99,1.1045211254112045E-4)
(100,1.1045211254112045E-4)
```

Show the lineage for RDD Ranks after 1, 2, and 3 loop iterations.

```
20/03/27 03:18:54 INFO io.SparkHadoopWriter: Job job_20200327031853_0062 committed.
(1) MapPartitionsRDD[61] at map at PageRank.scala:80 []
 |  ShuffledRDD[60] at reduceByKey at PageRank.scala:77 []
 +-(1) MapPartitionsRDD[59] at flatMap at PageRank.scala:71 []
    |  MapPartitionsRDD[58] at join at PageRank.scala:70 []
    |  MapPartitionsRDD[57] at join at PageRank.scala:70 []
    |  CoGroupedRDD[56] at join at PageRank.scala:70 []
    +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
    +-(1) MapPartitionsRDD[55] at map at PageRank.scala:80 []
       |  ShuffledRDD[54] at reduceByKey at PageRank.scala:77 []
       +-(1) MapPartitionsRDD[53] at flatMap at PageRank.scala:71 []
          |  MapPartitionsRDD[52] at join at PageRank.scala:70 []
          |  MapPartitionsRDD[51] at join at PageRank.scala:70 []
          |  CoGroupedRDD[50] at join at PageRank.scala:70 []
          +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
          +-(1) MapPartitionsRDD[49] at map at PageRank.scala:80 []
             |  ShuffledRDD[48] at reduceByKey at PageRank.scala:77 []
             +-(1) MapPartitionsRDD[47] at flatMap at PageRank.scala:71 []
                |  MapPartitionsRDD[46] at join at PageRank.scala:70 []
                |  MapPartitionsRDD[45] at join at PageRank.scala:70 []
                |  CoGroupedRDD[44] at join at PageRank.scala:70 []
                +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
                +-(1) MapPartitionsRDD[43] at map at PageRank.scala:80 []
                   |  ShuffledRDD[42] at reduceByKey at PageRank.scala:77 []
                   +-(1) MapPartitionsRDD[41] at flatMap at PageRank.scala:71 []
                      |  MapPartitionsRDD[40] at join at PageRank.scala:70 []
                      |  MapPartitionsRDD[39] at join at PageRank.scala:70 []
                      |  CoGroupedRDD[38] at join at PageRank.scala:70 []
                      +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
                      +-(1) MapPartitionsRDD[37] at map at PageRank.scala:80 []
                         |  ShuffledRDD[36] at reduceByKey at PageRank.scala:77 []
```

```
+-(1) MapPartitionsRDD[35] at flatMap at PageRank.scala:71 []
  |  MapPartitionsRDD[34] at join at PageRank.scala:70 []
  |  MapPartitionsRDD[33] at join at PageRank.scala:70 []
  |  CoGroupedRDD[32] at join at PageRank.scala:70 []
  +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
  +-(1) MapPartitionsRDD[31] at map at PageRank.scala:80 []
    |  ShuffledRDD[30] at reduceByKey at PageRank.scala:77 []
    +-(1) MapPartitionsRDD[29] at flatMap at PageRank.scala:71 []
      |  MapPartitionsRDD[28] at join at PageRank.scala:70 []
      |  MapPartitionsRDD[27] at join at PageRank.scala:70 []
      |  CoGroupedRDD[26] at join at PageRank.scala:70 []
      +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
      +-(1) MapPartitionsRDD[25] at map at PageRank.scala:80 []
        |  ShuffledRDD[24] at reduceByKey at PageRank.scala:77 []
        +-(1) MapPartitionsRDD[23] at flatMap at PageRank.scala:71 []
          |  MapPartitionsRDD[22] at join at PageRank.scala:70 []
          |  MapPartitionsRDD[21] at join at PageRank.scala:70 []
          |  CoGroupedRDD[20] at join at PageRank.scala:70 []
          +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
          +-(1) MapPartitionsRDD[19] at map at PageRank.scala:80 []
            |  ShuffledRDD[18] at reduceByKey at PageRank.scala:77 []
            +-(1) MapPartitionsRDD[17] at flatMap at PageRank.scala:71 []
              |  MapPartitionsRDD[16] at join at PageRank.scala:70 []
              |  MapPartitionsRDD[15] at join at PageRank.scala:70 []
              |  CoGroupedRDD[14] at join at PageRank.scala:70 []
              +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
              +-(1) MapPartitionsRDD[13] at map at PageRank.scala:80 []
                |  ShuffledRDD[12] at reduceByKey at PageRank.scala:77 []
                +-(1) MapPartitionsRDD[11] at flatMap at PageRank.scala:71 []
                  |  MapPartitionsRDD[10] at join at PageRank.scala:70 []
                  |  MapPartitionsRDD[9] at join at PageRank.scala:70 []
                  |  CoGroupedRDD[8] at join at PageRank.scala:70 []
                  +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66 []
                  +-(1) MapPartitionsRDD[7] at map at PageRank.scala:80 []
                    |  ShuffledRDD[6] at reduceByKey at PageRank.scala:77 []
                    +-(1) MapPartitionsRDD[5] at flatMap at PageRank.scala:71 []
                      |  MapPartitionsRDD[4] at join at PageRank.scala:70 []
                      |  MapPartitionsRDD[3] at join at PageRank.scala:70 []
                      |  CoGroupedRDD[2] at join at PageRank.scala:70 []
                      +-(1) ParallelCollectionRDD[0] at parallelize at PageRank.scala:66
[]
                      +-(1) ParallelCollectionRDD[1] at parallelize at PageRank.scala:67
[]
20/03/27 03:18:54 INFO spark.SparkContext: Invoking stop() from shutdown hook
```

Home work 4 Page Rank Manjit Ullal

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

The lineage above shows what operations are run by spark
spark uses a lazy execution model, so an action triggers a transformation

Report your observations: (1) Is Spark smart enough to figure out that it can re-use RDDs computed for an earlier action? (2) How do persist() and cache() change this behavior? (4 points)

1) if we do not persist, then spark will re compute the RDD even though it was used in earlier action, since there is no way for spark to know if the RDD is required in future
2) persist and cache allows us , not to re compute RDD that was already computed
cache uses default storage level but in persist we can mention the storage level

Page Rank in Map Reduce

pseudo-code

```
Mapper

      method map (nid n, node N)

            emit (n,N)
            p = pagerank
            emit(nid m, pagerank)

Reducer

      method reduce (nid m, [p1,p2,p3…..pn])
            node = null
            pg = 0
            for all n in counts [ p1,p2,p3…..pn] do
                  if isnode(n) then
                        node = p
                  else
                        if m equals 0 then
                              rank = p.pagerank
                              delta = delta + rank
                        else
                              pg = pg + p.pagerank
            m.pagerank = m.pagerank + delta/length(adjacency list)
            emit ( nid m, node m)
```

Home work 4 Page Rank Manjit Ullal

I. ***Describe briefly (1 paragraph) how you solved the dangling-page problem.***

The dangling page problem was solved by creating a dummy node with a neighbor of “0”. The logic is to add all number of all the dangling nodes using a global counter, and then redistribute the mass accumulated by these nodes back to other non-dangling nodes of the graph. The initial value of the page rank of dangling node is set to zero.

Running time for each cluster for k=1000 and 10 iterations:

	5 cheap machine	9 cheap machines
PageRank	22 mins	15 mins

Deliverable s

```
home folder:
total 36
-rw-rw-r--  1 manjit manjit 18909 Mar 27 04:06 Manjit-HW4.odt
drwxr-xr-x  8 manjit manjit  4096 Mar 27 04:18 Spark-PageRank
drwxr-xr-x 16 manjit manjit  4096 Mar 27 04:18 Hadoop-PageRank
drwxr-xr-x  2 manjit manjit  4096 Mar 27 04:19 logs
drwxr-xr-x  2 manjit manjit  4096 Mar 27 04:20 output
```

source code for Spark

```
manjit@ubuntu:~/Downloads/HW4/Spark-PageRank/src/main/scala/pg$ pwd
/home/manjit/Downloads/HW4/Spark-PageRank/src/main/scala/pg
manjit@ubuntu:~/Downloads/HW4/Spark-PageRank/src/main/scala/pg$ ls -ltr
total 4
-rw-r--r-- 1 manjit manjit 2803 Mar 27 03:29 PageRank.scala
```

source code for map reduce

```
manjit@ubuntu:~/Downloads/HW4/Hadoop-PageRank/src/main/java/pg$ pwd
/home/manjit/Downloads/HW4/Hadoop-PageRank/src/main/java/pg
manjit@ubuntu:~/Downloads/HW4/Hadoop-PageRank/src/main/java/pg$ ls -ltr
total 12
-rw-rw-r-- 1 manjit manjit 1123 Mar 27 01:37 graph.java
-rw-rw-rw- 1 manjit manjit 4106 Mar 27 04:03 pagerank.java
```