CS 6240- Large-scale Parallel Data Processing
Homework 4
Name: Srikanth Babu Mandru

Words used in document:

tuple._1 – value at first position in record
tuple._2 – value at second position in record

PAGERANK (Spark/Scala) Implementation:

Pseudo Code:

```
// create graphs table of form (node1, node2) and ranks table of form (node, pagerank)
graph_rdd ← load graphs table as RDD
ranks_rdd ← load ranks table as RDD
num_of_iters  ←10                              // set the number of iterations for pagerank update

// update pagerank iteratively
for (i ← 1 to num_of_iters) {
// Join both tables to get records of form (node1, (node2, pagerank))
    joined_rdd ← graph_rdd. join (ranks_rdd)

// list all the pairs (destination vertex, Pagerank).
// If dangling vertex is reached, add (dangling vertex, 0.0) and (0, 0.0)
    temp_rdd2 ← joined_rdd. flatMap (tuple => {
        if (tuple._1 % k == 1) :
                return ((tuple._1 , 0.0 ), tuple._2)
        else:
                return tuple._2 } )

// sum up the page rank for each destination vertex in an edge
    pagerank_sum_rdd ← temp_rdd2. groupByKey(). map (tuple => (tuple._1 , tuple._2.sum ))

// calculate pagerank lost due to dangling nodes using dummy node “0”
    lost_pagerank ← pagerank_sum_rdd. lookup(0).head

// distribute lost pagerank evenly to each node except dangling node "0" and update "ranks_rdd"
// for next iteration with pagerank computed using “pagerank formula”
    ranks_rdd ← pagerank_sum_rdd. map (tuple => {
        if (tuple._1 != 0 ) :
                alpha ← 0.15                                        // random jump factor
                dang_mass ← lost_pagerank/num_of_nodes              // pagerank at dangling nodes
                HJ_part ← alpha/num_of_nodes                        // hyperjump factor
                out_pr ← (tuple._1, HJ_part +  ((1 - alpha) * (tuple._2 + dang_mass)))
                return out_pr
        else: return (tuple._1 , 0.0) })
}                                                                   // End of FOR Loop
```

Operations that perform an action:

For PageRank computation, the following operations perform an action as they triggered job:

1. lookup
2. sum

Apart from the above operations, following operations in my program perform action which are used for output (does not account for Page Rank computation):

1. sortByKey
2. take

Final Page Ranks of vertices 0 to 100:

(0,0.0)
(1,1.588812573561084E-5)
(2,2.9391220523344347E-5)
(3,4.086705694684287E-5)
(4,5.061974152456418E-5)
(5,5.890776462131795E-5)
(6,6.595084287305437E-5)
(7,7.193573524791714E-5)
(8,7.702118669812162E-5)
(9,8.134213026403417E-5)
(10,8.501325886262294E-5)
(11,1.0470069929669521E-4)
(12,1.0470069929669521E-4)
(13,1.0470069929669521E-4)
(14,1.0470069929669521E-4)
(15,1.0470069929669521E-4)
(16,1.0470069929669521E-4)
(17,1.0470069929669521E-4)
(18,1.0470069929669521E-4)
(19,1.0470069929669521E-4)
(20,1.0470069929669521E-4)
(21,1.0470069929669521E-4)
(22,1.0470069929669521E-4)
(23,1.0470069929669521E-4)
(24,1.0470069929669521E-4)
(25,1.0470069929669521E-4)
(26,1.0470069929669521E-4)
(27,1.0470069929669521E-4)
(28,1.0470069929669521E-4)
(29,1.0470069929669521E-4)
(30,1.0470069929669521E-4)
(31,1.0470069929669521E-4)
(32,1.0470069929669521E-4)
(33,1.0470069929669521E-4)
(34,1.0470069929669521E-4)
(35,1.0470069929669521E-4)
(36,1.0470069929669521E-4)
(37,1.0470069929669521E-4)
(38,1.0470069929669521E-4)
(39,1.0470069929669521E-4)
(40,1.0470069929669521E-4)
(41,1.0470069929669521E-4)
(42,1.0470069929669521E-4)
(43,1.0470069929669521E-4)
(44,1.0470069929669521E-4)
(45,1.0470069929669521E-4)
(46,1.0470069929669521E-4)
(47,1.0470069929669521E-4)
(48,1.0470069929669521E-4)
(49,1.0470069929669521E-4)
(50,1.0470069929669521E-4)
(51,1.0470069929669521E-4)
(52,1.0470069929669521E-4)
(53,1.0470069929669521E-4)
(54,1.0470069929669521E-4)
(55,1.0470069929669521E-4)
(56,1.0470069929669521E-4)
(57,1.0470069929669521E-4)
(58,1.0470069929669521E-4)
(59,1.0470069929669521E-4)
(60,1.0470069929669521E-4)
(61,1.0470069929669521E-4)
(62,1.0470069929669521E-4)
(63,1.0470069929669521E-4)
(64,1.0470069929669521E-4)
(65,1.0470069929669521E-4)
(66,1.0470069929669521E-4)
(67,1.0470069929669521E-4)
(68,1.0470069929669521E-4)
(69,1.0470069929669521E-4)
(70,1.0470069929669521E-4)
(71,1.0470069929669521E-4)
(72,1.0470069929669521E-4)
(73,1.0470069929669521E-4)

(74,1.0470069929669521E-4)
(75,1.0470069929669521E-4)
(76,1.0470069929669521E-4)
(77,1.0470069929669521E-4)
(78,1.0470069929669521E-4)
(79,1.0470069929669521E-4)
(80,1.0470069929669521E-4)
(81,1.0470069929669521E-4)
(82,1.0470069929669521E-4)
(83,1.0470069929669521E-4)
(84,1.0470069929669521E-4)
(85,1.0470069929669521E-4)
(86,1.0470069929669521E-4)
(87,1.0470069929669521E-4)
(88,1.0470069929669521E-4)
(89,1.0470069929669521E-4)
(90,1.0470069929669521E-4)
(91,1.0470069929669521E-4)
(92,1.0470069929669521E-4)
(93,1.0470069929669521E-4)
(94,1.0470069929669521E-4)
(95,1.0470069929669521E-4)
(96,1.0470069929669521E-4)
(97,1.0470069929669521E-4)
(98,1.0470069929669521E-4)
(99,1.0470069929669521E-4)
(100,1.0470069929669521E-4)

Lineage for RDD ranks after 1 iteration:

```
Debug Info:
(4) MapPartitionsRDD[10] at map at PageRank.scala:77 []
 |  MapPartitionsRDD[7] at map at PageRank.scala:71 []
 |  ShuffledRDD[6] at groupByKey at PageRank.scala:71 []
 +-(4) MapPartitionsRDD[5] at flatMap at PageRank.scala:62 []
    |  MapPartitionsRDD[4] at join at PageRank.scala:59 []
    |  MapPartitionsRDD[3] at join at PageRank.scala:59 []
    |  CoGroupedRDD[2] at join at PageRank.scala:59 []
    +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:50 []
    |  |  CachedPartitions:4;MemorySize:351.6 KB;ExternalBlockStoreSize:0.0B;DiskSize:0.0B
    +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:52 []
```

Lineage for RDD ranks after 2 iterations:

```
Debug Info:
(4) MapPartitionsRDD[20] at map at PageRank.scala:77 []
 |  MapPartitionsRDD[17] at map at PageRank.scala:71 []
 |  ShuffledRDD[16] at groupByKey at PageRank.scala:71 []
 +-(4) MapPartitionsRDD[15] at flatMap at PageRank.scala:62 []
    |  MapPartitionsRDD[14] at join at PageRank.scala:59 []
    |  MapPartitionsRDD[13] at join at PageRank.scala:59 []
    |  CoGroupedRDD[12] at join at PageRank.scala:59 []
    +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:50 []
    |  |  CachedPartitions: 4; MemorySize: 351.6 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
    +-(4) MapPartitionsRDD[10] at map at PageRank.scala:77 []
       |  MapPartitionsRDD[7] at map at PageRank.scala:71 []
       |  ShuffledRDD[6] at groupByKey at PageRank.scala:71 []
       +-(4) MapPartitionsRDD[5] at flatMap at PageRank.scala:62 []
          |  MapPartitionsRDD[4] at join at PageRank.scala:59 []
          |  MapPartitionsRDD[3] at join at PageRank.scala:59 []
          |  CoGroupedRDD[2] at join at PageRank.scala:59 []
          +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:50 []
          |  |  CachedPartitions: 4; MemorySize: 351.6 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
          +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:52 []
```

Lineage for RDD ranks after 3 iterations:

```
Debug Info :
(4) MapPartitionsRDD[30] at map at PageRank.scala:77 []
 |  MapPartitionsRDD[27] at map at PageRank.scala:71 []
 |  ShuffledRDD[26] at groupByKey at PageRank.scala:71 []
 +-(4) MapPartitionsRDD[25] at flatMap at PageRank.scala:62 []
    |  MapPartitionsRDD[24] at join at PageRank.scala:59 []
    |  MapPartitionsRDD[23] at join at PageRank.scala:59 []
    |  CoGroupedRDD[22] at join at PageRank.scala:59 []
    +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:50 []
    | |      CachedPartitions: 4; MemorySize: 351.6 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
    +-(4) MapPartitionsRDD[20] at map at PageRank.scala:77 []
       |  MapPartitionsRDD[17] at map at PageRank.scala:71 []
       |  ShuffledRDD[16] at groupByKey at PageRank.scala:71 []
       +-(4) MapPartitionsRDD[15] at flatMap at PageRank.scala:62 []
          |  MapPartitionsRDD[14] at join at PageRank.scala:59 []
          |  MapPartitionsRDD[13] at join at PageRank.scala:59 []
          |  CoGroupedRDD[12] at join at PageRank.scala:59 []
          +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:50 []
          | |      CachedPartitions: 4; MemorySize: 351.6 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
          +-(4) MapPartitionsRDD[10] at map at PageRank.scala:77 []
             |  MapPartitionsRDD[7] at map at PageRank.scala:71 []
             |  ShuffledRDD[6] at groupByKey at PageRank.scala:71 []
             +-(4) MapPartitionsRDD[5] at flatMap at PageRank.scala:62 []
                |  MapPartitionsRDD[4] at join at PageRank.scala:59 []
                |  MapPartitionsRDD[3] at join at PageRank.scala:59 []
                |  CoGroupedRDD[2] at join at PageRank.scala:59 []
                +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:50 []
                | |      CachedPartitions:4;MemorySize:351.6KB;ExternalBlockStoreSize:0.0B;DiskSize: 0.0 B
                +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:52 []
```

Discussion:

At first, I have created both the graph table and ranks table where graph table consists of edges in graph as record and ranks table consist of pagerank for each page or node in graph. Joining both tables will result in the (source vertex, (destination vertex, source pagerank)) pairs. Now, on this pairs, for each (destination vertex, source pagerank) sum all the source vertices page ranks through group by of “destination vertex” which results in (vertex, pagerank sum). Then pagerank sum of vertex “0” gives total pagerank being lost due to dangling nodes. Finally, use the formula of page rank to include “random jump factor” and “distributed PageRank mass” and replace pagerank sum with new value of page rank computed using formula. This is final output of current PageRank iteration and is passed as input to next iteration. Thus, the PageRank program provided as in Pseudo code above computes the PageRank.

In terms of jobs execution, first the “lookup” triggered the job to compute the total page rank being lost. “sum” triggers another job to compute sum of page ranks for all the vertices. These results will be then used for further pagerank computation and written to the output files.

Observations:

1. Yes, Spark is smart enough so that it can re-use RDDs computed for an earlier action.
2. persist () and cache () helps in saving the previously computed RDDs so that those intermediate results can be used in further iterations.

PAGERANK (MapReduce/Java) Implementation:

Brief explanation:

I have used same algorithm stated in the "Data-Intensive Text Processing with MapReduce" textbook. I have used dummy node ("0") to collect all the dangling nodes pagerank and MapReduce Job Counter to keep it for next iteration. Computed the PageRank as per the formula given in textbook, which is as follows:

$$p' = \alpha \left(\frac{1}{|G|}\right) + (1 - \alpha)\left(\frac{M}{|G|} + p\right)$$

Pseudo Code:

```
class MAPPER

    method MAP (nid n, node N)
        p ← N.PAGERANK / |N.ADJACENCYLIST|
        EMIT (nid n, N)

        for all nodeid m ∈ N.ADJACENCYLIST do
            EMIT (nid m, pagerank p)

class REDUCER

    method SETUP ()
        alpha ← 0.15                                    // set random jump factor
    // get the total number of nodes and dangling nodes pagerank from previous MR job
        total_nodes ← context.getCounter("TOTAL_NODES")
        dangling_pagerank ← context.getCounter("DANGLING_MASS")

    method REDUCE (nid m, pageranks [p1, p2, p3, …. ])
        M ← ∅
        s ← 0
        for all p ∈ pageranks [p1, p2, p3, …. ] do
            if IsNode(p) then
                M ← p
            else
                s ← s + p

        hyperjump_factor ← alpha / total_nodes
        dangling_contribution ← dangling_pagerank / total_nodes
        hyperlink_term ← (1 - alpha) * (s + dangling_contribution )
        M.PAGERANK ← hyperjump_factor + hyperlink_term
```

```
        if (key != 0 ):
                EMIT(nid m, node M)
        else:
// since dummy vertex ("0") collects all the dangling nodes mass, assign pagerank of "0" to the
// counter so that it can be used in next iteration
                context.Counter("DANGLING_MASS").increment(s)
```

Solution for dangling-page problem:

To solve the problem of dangling nodes, I have used the dummy vertex "0" and added an edge "0.0" to input file with edges. So, from mapper, "0" is emitted for all the dangling nodes with pagerank importance of dangling nodes. In reducer, if I get to the vertex "0", then I sum all the pagerank values and assign that value to global counter which will be used in the next iteration while computing the pagerank on reducer side.

To check Correctness of program:

To know the total page rank lost in current iteration, total page rank of all vertices and iteration number, I have logged that information in log files.

NLineInputFormat:

Below is screenshot of NLineInputFormat included in MapReduce Job. "numLines" set to 50000 since "(1000000 / 50000) = 20".

```
// FileInputFormat.setInputPaths(job, in);
NLineInputFormat.setInputPaths(job, in);
NLineInputFormat.setNumLinesPerSplit(job, numLines: 50000);
FileOutputFormat.setOutputPath(job, jobOutputPath);
```

Running time for each cluster:

Runtime is captured from the "controller.txt" file which included beside "syslog" files for respective cluster run.

Cluster Type	Runtime
Small cluster (1 master and 4 workers)	516 seconds
Large cluster (1 master and 8 workers)	496 seconds

Input File for MapReduce job:

I have used python programming language to create the input file. I included python file in the homework folder at "HW4/FileCreate.ipynb"

Log files path:

For Spark project: logs are provided in "HW4/spark_logs/"
For MapReduce project: logs are provided in "HW4/mapreduce_logs/"

Output files path:

For Spark project: outputs are provided in "HW4/spark_outputs/"

For MapReduce project: outputs are provided in "HW4/mapreduce_outputs/"

Note: For MapReduce Job, output for each iteration of MapReduce Job is placed in respective number "#" folder that is output of 10_{th} iteration is written to "10" folder of small or large cluster run output folders.

Projects path:

Spark project: "HW4/Spark-PageRank/"

MapReduce project: "HW4/MapReduce-PageRank/"

Readme path: (for execution steps or procedure)

For Spark project: "HW4/Spark-PageRank/Readme.txt"

For MapReduce project: "HW4/MapReduce-PageRank/Readme.txt"

Report path:
HW4/Srikanth_Mandru_HW4.pdf