**CS6240 LARGE SCALE DATA PROCESSING**

**HOMEWORK 4**

**SNEH GURDASANI**

**Pseudocode for PageRank in Spark (Scala):**

1. Form two RDDs:

* Graph RDD - This RDD contains the synthetic graph. The nodes are of the form: (from\_node, to\_node). Since we have only one connection, for each key(from\_node), we get only one value(to\_node).
* Rank RDD – This RDD is of the form – (from\_node, pagerank), where from\_node is obtained from the list of nodes.

2. Compute the **rank** to be 1.0/numOfNodes .

3. For all kth nodes (multiples of k) of Graph RDD, put the node (from\_node, 0). For all ‘from’ nodes of Graph RDD, put (from\_node , rank) into the Rank RDD except for node 0, put (0 , 0).

4. **Join** the two RDDs and form a new RDD ‘join\_rdd’. So, you get (from\_node, to\_node, pagerank) in the ‘join\_rdd’.

5. Now, transfer the PageRank value(**rank**) to every ‘to\_node’.

6. Once we are done with one iteration, we can add the (dummy-vertex(‘0’)’s Pagerank value/number of Nodes) to all the other nodes’ Pagerank value.

7. Repeat from step 3 iterating 10 times.

**Actions used:**

1. Reduce - ReduceByKey internally uses Reduce action. This is to sum up the values for different node values.

2. Lookup – To get the Pagerank value of node 0.

3. take – Take the Pagerank values for node 1 to 100.

**RDD Lineage:**

**Without persist():**

**Iteration 1:**

A screenshot of a cell phone

Description automatically generated

**Iterations 2:**

A close up of text on a black background

Description automatically generated

**Iterations 3:**

A screenshot of a cell phone

Description automatically generated

A close up of text on a black background

Description automatically generated

**Using persist():**

**Iteration 1:**

A screenshot of a cell phone

Description automatically generated

**Iteration 2:**

A close up of text on a black background

Description automatically generated

**Iteration 3:**

A screenshot of a cell phone

Description automatically generated

A screenshot of text

Description automatically generated

**ANS:**  I determined what was executed based on the DebugString() method. So, this method gives you the transformations and actions applied to the RDD. So, three operations are being performed: Join, Shuffle(and coalesce(describes the number of partitions)) and Parallelize.

**ANS:**

(1) Yes, Spark is smart enough to figure out that it can re-use RDDs computed for earlier action as can be seen in the lineage found with or without persist().

(2) When we use persist() or cache(), each node in the RDD stores the partitions into memory which can be used for future actions. Caching allows future actions to be faster.

**PageRank in MapReduce:**

1. I implemented PageRank similar to the algorithm in the book.

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, p)

class Reducer  
method Reduce(nid m, [p1, p2, . . .])

M←∅  
 for all p ∈ counts [p1,p2,...] do

if IsNode(p) then

M ← p

else

s ← s + p

M.PageRank ← s

Emit(nid m, node M )

A global counter is maintained to counts the delta value and the number of nodes after each iteration . Also, I limit the number of input lines to be read to 32000 to ensure at least 20 Map tasks.

2. I took the dummy-vertex approach. Here, I transferred the PageRank for every kth node to the dummy- vertex(0). At the end of each iteration, I transferred the PageRank value of dummy vertex to all the nodes.