**CS6240**-**Assignment 4**

**FALL 2020**

**Rahul Pandey**

**GitHub:**

[**https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs**](https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs) **(MR)**

[**https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs**](https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs) **(Spark)**

**PageRank Spark:** [**https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/blob/master/src/main/scala/pageRank/PageRankMain.scala**](https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/blob/master/src/main/scala/pageRank/PageRankMain.scala)

**PageRank MR code:** [**https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/blob/master/src/main/java/pageRank/PageRankFromSyntheticData.java**](https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/blob/master/src/main/java/pageRank/PageRankFromSyntheticData.java)

**PageRank in Spark (40 points total):**

The graph is of form (page1, page2) which signifies, page1 has outlink to page2. The another table generated on the fly is the pageRank table, which is of form (pageId, pageRank).

**PseudoCode:**

SparkPageRank(){

k = first\_Argument\_via\_CommandLine (100)

start = 1

pairV1V2 = new ListBuffer[Row]

pairEdgePageRank = new ListBuffer[Row]

pairEdgePageRank += Row(0, 0.0)

//Creating Graph and RDD on the fly

while (start <= k \* k) {

pairEdgePageRank += Row(start, 1.0 / (k \* k)) *//Defaulting pageRank for each page to 1/k\*k*

// Fill pairV1V2 with dummy

if (start % k == 0) { *//Creating the Dummy node at the end of each dangling page*

pairV1V2 += Row(start, 0)

} else {

pairV1V2 += Row(start, start + 1) *//Connecting pages 1->2->3-4 etc.*

}

start = start + 1

}

//args(2)

iters = if (args.length > 1) args(1).toInt else 10 *//If number of iteration is not passed default it to 10*

*//Convert pageRank and V1V2 pair to RDD*

pairV1V2RDD = sc.parallelize(pairV1V2) //Converting V1V2 Pair graph to RDD

pairEdgePageRankRDD = sc.parallelize(pairEdgePageRank) //Converting pagerank per page list to RDD

rddGraph = pairV1V2RDD.map(row => (row(0), List(row(1))))

.reduceByKey(\_++\_)

.cache() *//Cache the graph in the memory*

*//Converting to RDD(pageId, pageRank)*

rddPageRank = pairEdgePageRankRDD.map(row => (row(0), row(1).toString.toDouble))

.partitionBy(rddGraph.partitioner.get) //using same partition as rddGraph

for (i <- 1 to iters) {

*//Joining RDD’s*

temp = rddGraph.join(rddPageRank)

*//Calculating OutLink contribution*

contribs = rddGraph.join(rddPageRank).flatMap {

case (vertex, (adjNodes, pageRankOfVertex)) => val totalOutLinks = adjNodes.size

*//Distributing vertex pagerank over its adjacency list*

adjNodes.flatMap(url => List((url, pageRankOfVertex / totalOutLinks), (vertex, 0.0))) *//Pass vertex to keep track of originating vertex, adding 0.0 for vertex with no inlinks*

}

contribsMain = contribs.reduceByKey(\_+\_) //Sum pageRanks

//Get dangling page mass using the lookup for dummy node

val danglingMass = contribsMain.lookup(0).head

rddPageRank = contribsMain.map { //value is sum of pageRank of all pages adj to vertex

case (vertex, value) => if (vertex == 0) {

(vertex, 0.0)

} else { // Calculating pagerank

(vertex, 0.15 \* (1.0 / (k \* k)) + 0.85 \* (value + danglingMass \* (1.0 / (k \* k)))) //Distribute dangling pages mass

}

}.sortBy(\_.\_1.toString.toInt)

}

//args(2)

logger.info(rddPageRank.toDebugString)

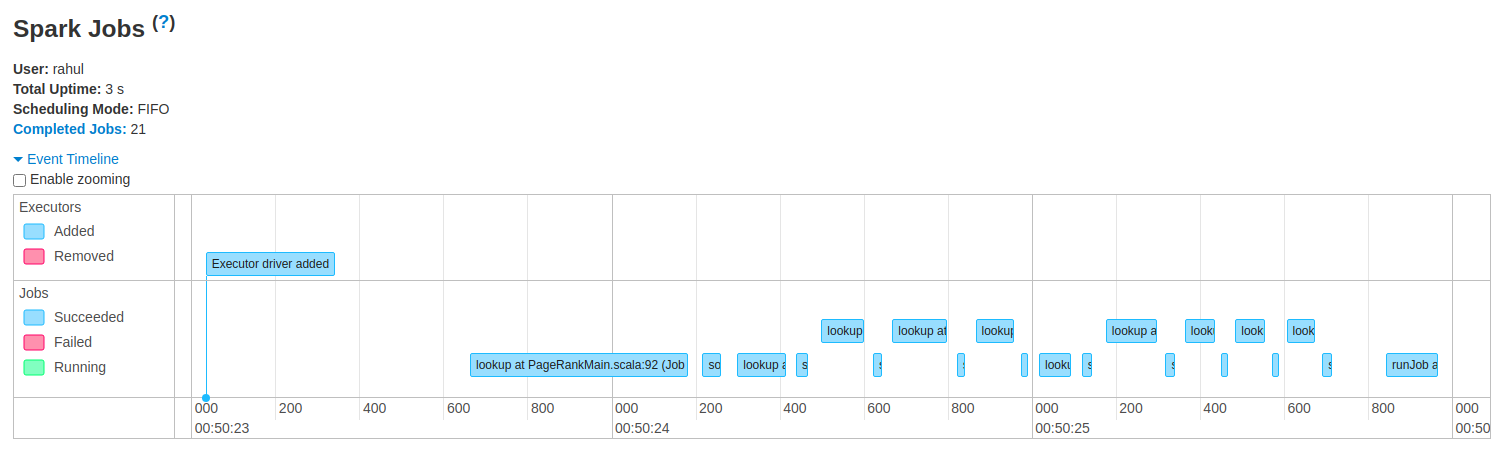
rddPageRank.saveAsTextFile(args(2)) //Save data to local storage

}

**Which of the operations in your program perform an action ?**

The operations sortBy, Lookup and saveAsTextFile() triggers an actions. This can be verified using the below screenshot from Spark UI visualization at:

<http://localhost:18080/history/local-1604728223014/jobs/>

****

The above result is for total of 10 iteration for k =10. We can see there are 10 lookup() and corresponding sortBy() call. We can see at the end the even around 800 is a write to local storage and is triggered by saveAsTextFile()

**Program for 10 iterations for k=100, and output:**

**Output:**

<https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/tree/master/localOutputK100>

**Logs:** <https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/blob/maste>[r/logs/SparkLocal\_K100/K100\_LocalRun.txt](https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/blob/master/logs/SparkLocal_K100/K100_LocalRun.txt)

**Show the lineage for RDD Ranks :**

*After 1 loop iterations:*

2020-11-07 01:05:41 INFO root:98 - (4) MapPartitionsRDD[19] at sortBy at PageRankMain.scala:94 []

| ShuffledRDD[18] at sortBy at PageRankMain.scala:94 []

+-(4) MapPartitionsRDD[15] at sortBy at PageRankMain.scala:94 []

| MapPartitionsRDD[14] at map at PageRankMain.scala:88 []

| ShuffledRDD[13] at reduceByKey at PageRankMain.scala:83 []

+-(4) MapPartitionsRDD[12] at flatMap at PageRankMain.scala:78 []

| MapPartitionsRDD[11] at join at PageRankMain.scala:78 []

| MapPartitionsRDD[10] at join at PageRankMain.scala:78 []

| CoGroupedRDD[9] at join at PageRankMain.scala:78 []

| ShuffledRDD[3] at reduceByKey at PageRankMain.scala:67 []

| CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B

+-(4) MapPartitionsRDD[2] at map at PageRankMain.scala:66 []

| ParallelCollectionRDD[0] at parallelize at PageRankMain.scala:63 []

| ShuffledRDD[5] at partitionBy at PageRankMain.scala:71 []

+-(4) MapPartitionsRDD[4] at map at PageRankMain.scala:70 []

| ParallelCollectionRDD[1] at parallelize at PageRankMain.scala:64 []

*After 2 loop iterations:*

2020-11-07 01:07:00 INFO root:98 - (4) MapPartitionsRDD[33] at sortBy at PageRankMain.scala:94 []

| ShuffledRDD[32] at sortBy at PageRankMain.scala:94 []

+-(4) MapPartitionsRDD[29] at sortBy at PageRankMain.scala:94 []

| MapPartitionsRDD[28] at map at PageRankMain.scala:88 []

| ShuffledRDD[27] at reduceByKey at PageRankMain.scala:83 []

+-(4) MapPartitionsRDD[26] at flatMap at PageRankMain.scala:78 []

| MapPartitionsRDD[25] at join at PageRankMain.scala:78 []

| MapPartitionsRDD[24] at join at PageRankMain.scala:78 []

| CoGroupedRDD[23] at join at PageRankMain.scala:78 []

| ShuffledRDD[3] at reduceByKey at PageRankMain.scala:67 []

| CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B

+-(4) MapPartitionsRDD[2] at map at PageRankMain.scala:66 []

| ParallelCollectionRDD[0] at parallelize at PageRankMain.scala:63 []

+-(4) MapPartitionsRDD[19] at sortBy at PageRankMain.scala:94 []

| ShuffledRDD[18] at sortBy at PageRankMain.scala:94 []

+-(4) MapPartitionsRDD[15] at sortBy at PageRankMain.scala:94 []

| MapPartitionsRDD[14] at map at PageRankMain.scala:88 []

| ShuffledRDD[13] at reduceByKey at PageRankMain.scala:83 []

+-(4) MapPartitionsRDD[12] at flatMap at PageRankMain.scala:78 []

| MapPartitionsRDD[11] at join at PageRankMain.scala:78 []

| MapPartitionsRDD[10] at join at PageRankMain.scala:78 []

| CoGroupedRDD[9] at join at PageRankMain.scala:78 []

| ShuffledRDD[3] at reduceByKey at PageRankMain.scala:67 []

| CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B

+-(4) MapPartitionsRDD[2] at map at PageRankMain.scala:66 []

| ParallelCollectionRDD[0] at parallelize at PageRankMain.scala:63 []

| ShuffledRDD[5] at partitionBy at PageRankMain.scala:71 []

+-(4) MapPartitionsRDD[4] at map at PageRankMain.scala:70 []

| ParallelCollectionRDD[1] at parallelize at PageRankMain.scala:64 []

*After 3 loop iterations:*

2020-11-07 01:08:29 INFO root:98 - (4) MapPartitionsRDD[47] at sortBy at PageRankMain.scala:94 []

| ShuffledRDD[46] at sortBy at PageRankMain.scala:94 []

+-(4) MapPartitionsRDD[43] at sortBy at PageRankMain.scala:94 []

| MapPartitionsRDD[42] at map at PageRankMain.scala:88 []

| ShuffledRDD[41] at reduceByKey at PageRankMain.scala:83 []

+-(4) MapPartitionsRDD[40] at flatMap at PageRankMain.scala:78 []

| MapPartitionsRDD[39] at join at PageRankMain.scala:78 []

| MapPartitionsRDD[38] at join at PageRankMain.scala:78 []

| CoGroupedRDD[37] at join at PageRankMain.scala:78 []

| ShuffledRDD[3] at reduceByKey at PageRankMain.scala:67 []

| CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B

+-(4) MapPartitionsRDD[2] at map at PageRankMain.scala:66 []

| ParallelCollectionRDD[0] at parallelize at PageRankMain.scala:63 []

+-(4) MapPartitionsRDD[33] at sortBy at PageRankMain.scala:94 []

| ShuffledRDD[32] at sortBy at PageRankMain.scala:94 []

+-(4) MapPartitionsRDD[29] at sortBy at PageRankMain.scala:94 []

| MapPartitionsRDD[28] at map at PageRankMain.scala:88 []

| ShuffledRDD[27] at reduceByKey at PageRankMain.scala:83 []

+-(4) MapPartitionsRDD[26] at flatMap at PageRankMain.scala:78 []

| MapPartitionsRDD[25] at join at PageRankMain.scala:78 []

| MapPartitionsRDD[24] at join at PageRankMain.scala:78 []

| CoGroupedRDD[23] at join at PageRankMain.scala:78 []

| ShuffledRDD[3] at reduceByKey at PageRankMain.scala:67 []

| CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B

+-(4) MapPartitionsRDD[2] at map at PageRankMain.scala:66 []

| ParallelCollectionRDD[0] at parallelize at PageRankMain.scala:63 []

+-(4) MapPartitionsRDD[19] at sortBy at PageRankMain.scala:94 []

| ShuffledRDD[18] at sortBy at PageRankMain.scala:94 []

+-(4) MapPartitionsRDD[15] at sortBy at PageRankMain.scala:94 []

| MapPartitionsRDD[14] at map at PageRankMain.scala:88 []

| ShuffledRDD[13] at reduceByKey at PageRankMain.scala:83 []

+-(4) MapPartitionsRDD[12] at flatMap at PageRankMain.scala:78 []

| MapPartitionsRDD[11] at join at PageRankMain.scala:78 []

| MapPartitionsRDD[10] at join at PageRankMain.scala:78 []

| CoGroupedRDD[9] at join at PageRankMain.scala:78 []

| ShuffledRDD[3] at reduceByKey at PageRankMain.scala:67 []

| CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B

+-(4) MapPartitionsRDD[2] at map at PageRankMain.scala:66 []

| ParallelCollectionRDD[0] at parallelize at PageRankMain.scala:63 []

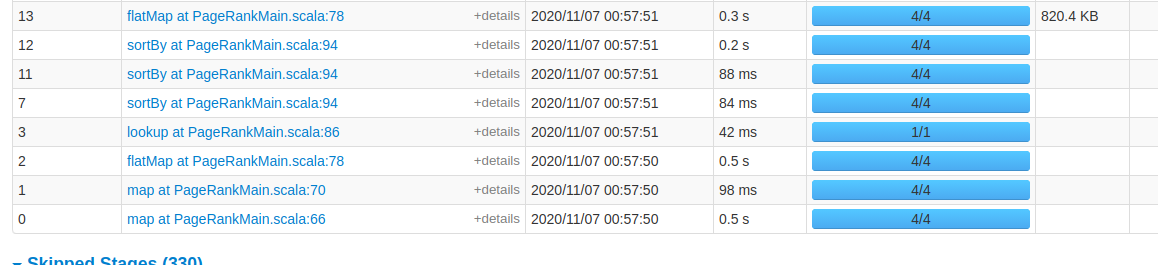
| ShuffledRDD[5] at partitionBy at PageRankMain.scala:71 []

+-(4) MapPartitionsRDD[4] at map at PageRankMain.scala:70 []

| ParallelCollectionRDD[1] at parallelize at PageRankMain.scala:64 []

**How you determined what was actually executed by a job triggered by your program:**

This is evident by looking at Pagerank job execution in Spark UI. Looking at various stages executed in the UI reveals what happened under the hood.



**Startup:**

It can be seen clearly the first stage points to the map at line 66 of the program. That is when we try to create the graph RDD:

*val rddGraph = pairV1V2RDD.map(row => (row(0), List(row(1))))*

*.reduceByKey(\_++\_)*

*.cache() //Cache the graph in the memory*

The second stage (1) is similar creation of pageRankRDD using map.

Then in Stage(2) we have a flatMap that represents the join between these two tables and calculating the contribution.

Then finally we have a lookup for getting the dangling mass from dummy node.

This is followed by the sortBy() which sorts the data based on the key.

**Iterations:**

The next set of execution include the same set of steps I.e:

* flatMap → For joining the two RDD’s.
* Lookup to get the dangling mass.
* SortBy to sort the data based on key as received from the map job.

**Sequence followed**:

sortBy at PageRankMain.scala:94

sortBy at PageRankMain.scala:94

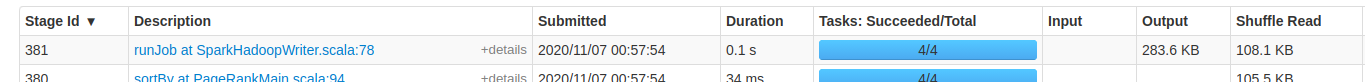
sortBy at PageRankMain.scala:94

lookup at PageRankMain.scala:86

flatMap at PageRankMain.scala:78

**Last stage:**

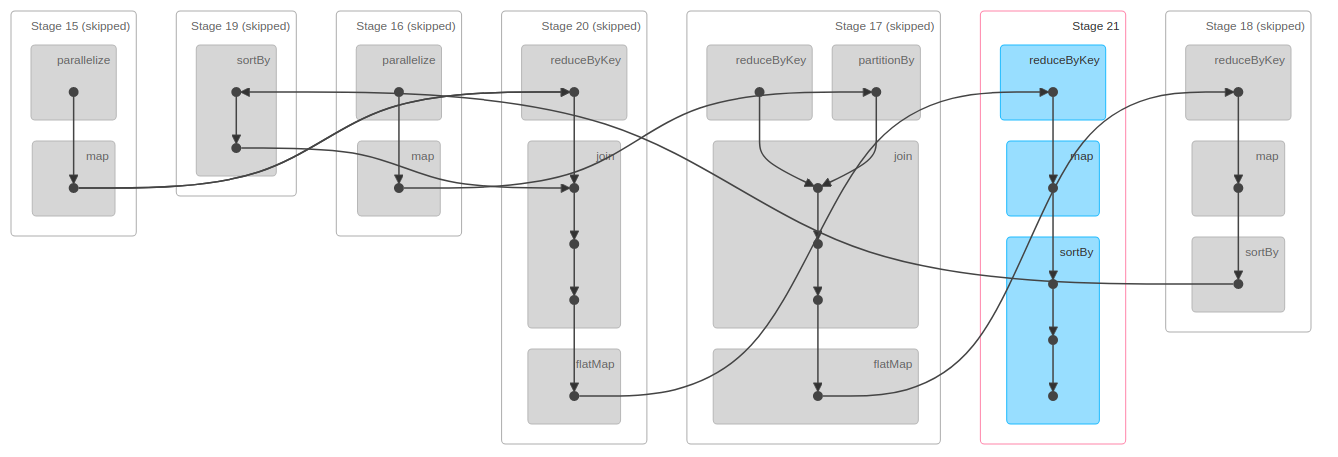
The last stage is saveAsTextFile(), this is evident from the last job we see in the completed stages:



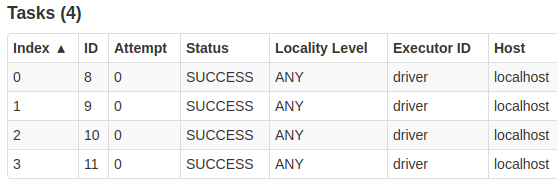
**Report your observations:**

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

Yes, The Spark is smart enough to not recompute RDD’s from previous action and use it to improve the performance. As seen in below DAG visualization the previous steps were skipped as they were served from the memory.



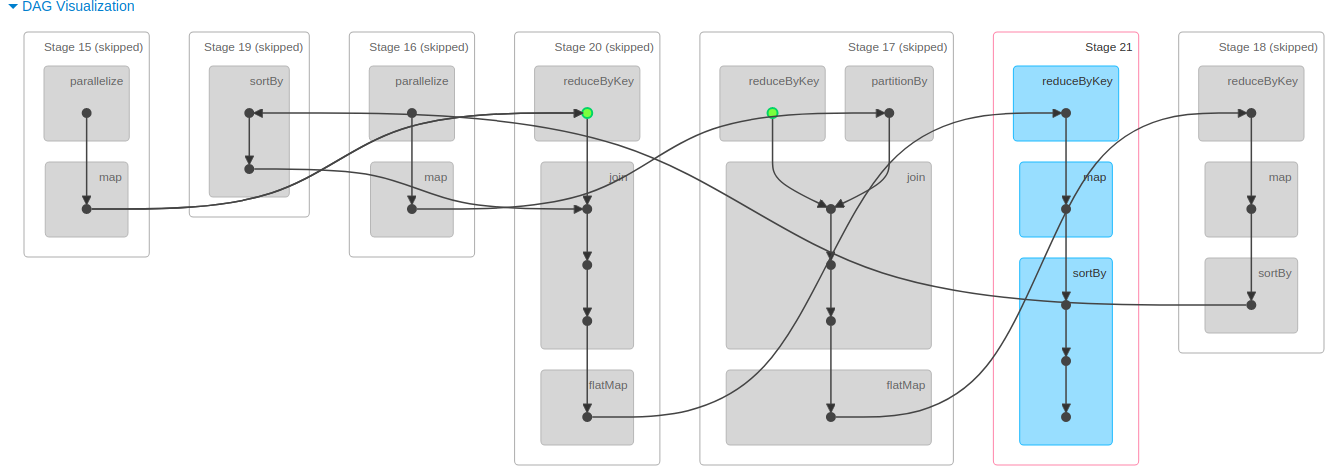
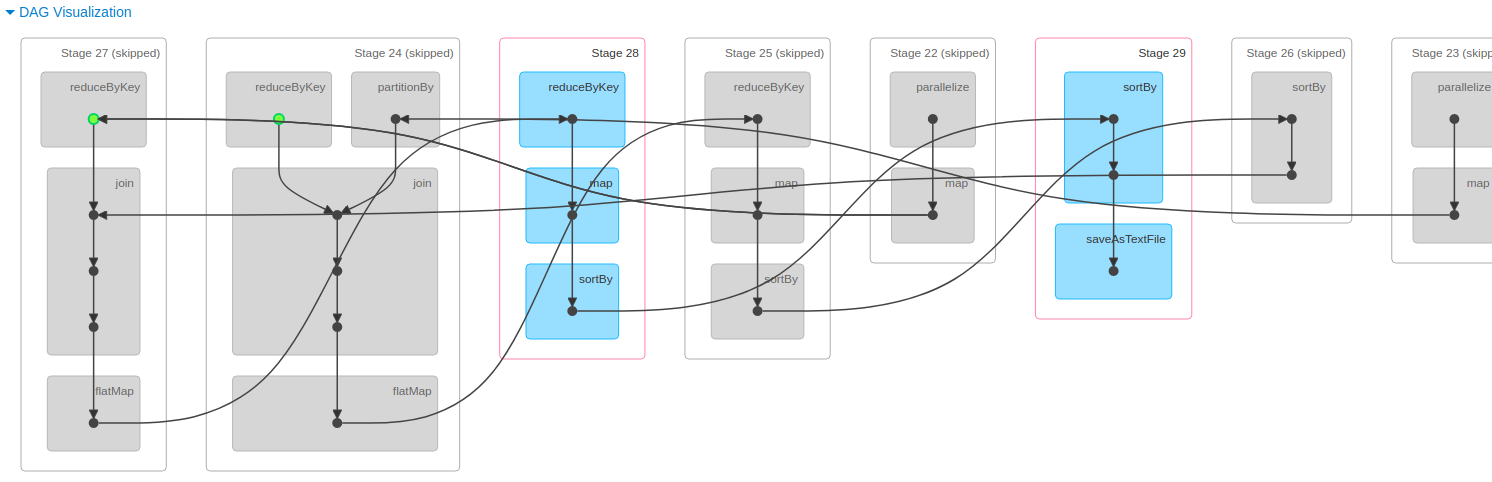
Looking at process status the locality level is ANY as we have not explicitly asked spark to cache() or persist() this and it is just one of the performance optimization it is doing in the background.

****

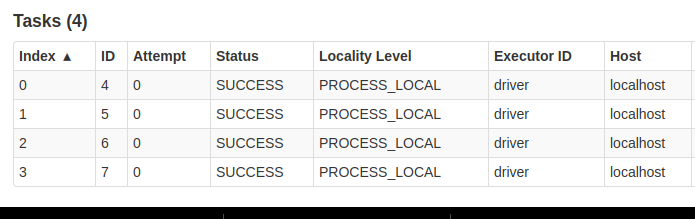
**(2) How do persist() and cache() change this behavior?**

By using cache() and persist() we are telling the spark explicitly to store the data in cache and

during further processing serve it from the memory. Though the locality level changes to **PROCESS\_LOCAL** from **ANY**



As seen from the above DAG visualization there are several steps that were skipped as it was served from the memory. The green dot in the image show the RDD was cached.



**PageRank in MapReduce (20 points total)**

The input to the program is generated beforehand and each row in the input looks like:

**pageId1 pageId2, pageRank** e.g : *1 2,0.00010*

*The input is generates using a separate program present within the code.*

*Here pageId1 has a outlink to pageId2 and pageRank is the pageRank for pageId1*

The code is run for 10 iterations and k=1000 where each job output is input to the next as we keep updating the pagerank value in each iteration. Also, we have global counters **DELTA\_SUM** and **PAGE\_RANK\_SUM** to keep track of dangling mass sum that will be distributed over nodes and pagerank sum to ensure the final pagerank sum is close to 1.

**PsuedoCode:**

//Custom writable class that represents the node in the graph

class **PageNode** {

id = -1

pageRank = -1

adjList = ""

PageNode() {

}

//Constructor to create

PageNode(int id, double pageRank, String adjList) {

id = id

pageRank = pageRank

adjList = adjList

}

public void write(DataOutput dataOutput) throws IOException {

dataOutput.writeInt(id)

dataOutput.writeUTF(adjList)

dataOutput.writeDouble(pageRank)

}

//Readfields

public void readFields(DataInput dataInput) throws IOException {

id = dataInput.readInt()

adjList = dataInput.readUTF()

pageRank = dataInput.readDouble()

}

//String value used to write to output

public String toString() {

return "" + adjList + "," + this.pageRank

}

}

//Global counter

**enum PageRankGlobalCounter** {

PAGE\_RANK\_SUM, //Keep track of page rank sum for the last job

DELTA\_SUM // used to calculate dangling mass sum

}

class **PageRankFromSyntheticData** {

k = 1000;

class **PageRankMapper** {

totalNumberOfPages

delta

setup(context) {

*// get total number of pages effectively |V|*

totalNumberOfPages = getFromJobConfig("totalNumberOfPages"))

// Get delta from previous iteration as passed from previous run

delta = getFromJobConfig("delta"))

}

**map**(key, value, context) {

*// separate pagerank data from graph data*

graphAndPageRank[] = value.toString().split(",")

*//For input vertex X in row: X Y*

inputSplit[] = graphAndPageRank[0].split(" ")

*//Distribute dangling mass value*

calculatedPageRank = graphAndPageRank[1] + (0.85) \* delta / totalNumberOfPages

PageNode pageNode = PageNode(inputSplit[0], calculatedPageRank, (inputSplit.length > 1 ? inputSplit[1] : ""))

adjSize = inputSplit.length - 1

emit(new Text(pageNode.id, pageNode)

if (adjSize > 0) {

pgRnkToDistribute = ((calculatedPageRank) / (adjSize))

for (i = 1; i < inputSplit.length; i++) {

*//emit node with distributed page rank for each vertex*

emit(inputSplit[i]), PageNode(-1, pgRnkToDistribute, ""))

}

}

}

}

class **PageRankReducer** {

totalNumberOfPages = getFromJobConfiguration("totalNumberOfPages"))

**reduce**(key, Iterable<PageNode> values, context){

double sum = 0.0

PageNode newNode = new PageNode()

for (PageNode node <- values) {

if (node.id != -1) { *//The vertex was found, recover graph*

newNode = PageNode(node.id, node.pageRank, node.adjList)

} else {

sum += node.pageRank; *//Sum inlink contribution*

}

}

if (key!= 0) //If not the dummyNode and not null

newNode.pageRank = (0.15) \* (1.0) / (totalNumberOfPages) + (0.85 \* sum)

else

newNode.pageRank = 0.0

// emit(id adjList, pageRank)

emit(key, newNode)

//If dummyNode, pass pageRank to next node

if (newNode.id == 0) { //

getGlobalCounter(DELTA\_SUM).setValue((sum))

}

}

}

class **LastJobMapper** {

totalNumberOfPages =getFromJobConfiguration("totalNumberOfPages"))

delta = getFromJobConfiguration("delta"))

**map**(key, value,context) {

*// retrieve graph and pagerank data*

graphAndPageRank[] = value.toString().split(",")

if (graphAndPageRank.length != 2)

return

*//For input vertex X in row: X Y*

inputSplit[] = graphAndPageRank[0].split(" ")

PageNode pageNode = PageNode(inputSplit[0], 0.0, (inputSplit.length > 1 ? inputSplit[1] : ""))

*//If not dummy node, distribute dangling mass*

if (pageNode.id != 0) {

pageNode.pageRank = graphAndPageRank[1] + ((0.85 \* delta) / (totalNumberOfPages))

}

emit(pageNode.id, pageNode)

*// Sum pagerank for last ob to ensure it is close to 1*

getGlobalCounter(PAGE\_RANK\_SUM).increment((pageNode.pageRank))

}

}

DriverCode(input\_CLI) {

conf = getConf()

iterationNum = 1

delta = 0.0

**//Job 0**

Job job = Job.getInstance(conf, "PageRank MR: " + iterationNum)

jobConf = job.getConfiguration()

jobConf.set("delta", "" + delta) ***// set initial delta value***

jobConf.set("totalNumberOfPages", "" + k \* k) ***// set total\_pages to k\*k value***

job.setMapperClass(PageRankMapper.class)

job.setReducerClass(PageRankReducer.class)

*//For 20 map tasks, using NLINEINPUTFORMAT*

job.setInputFormatClass(NLineInputFormat.class)

job.setOutputKeyClass(Text.class)

job.setOutputValueClass(PageNode.class)

*//Setting input and output path from console*

NLineInputFormat.addInputPath(job, new Path(args[0]))

jobConf.setInt("mapreduce.input.lineinputformat.linespermap", k \* k / 20)

FileOutputFormat.setOutputPath(job, new Path(args[1] + "/Job" + iterationNum))

job.waitForCompletion(true)

iterationNum++ // Increase iteration

**//Next set of Jobs (Each job output is passed to next job as input)**

**//The delta is read for each previous job as currentDelta and passed to next Job**

while (iterationNum <= 10) {

currDelta = getGlobalCounter(DELTA\_SUM) **// previous job delta**

conf = getConf()

job = Job.getInstance(conf, "PageRankMR: " + iterationNum)

jobConf = job.getConfiguration()

jobConf.set("mapreduce.output.textoutputformat.separator", " ")

job.setMapperClass(PageRankMapper.class)

job.setReducerClass(PageRankReducer.class)

job.setInputFormatClass(NLineInputFormat.class)

job.setOutputKeyClass(Text.class)

job.setOutputValueClass(PageNode.class)

**//Setting global accessible data for job**

jobConf.set("delta", "" + currDelta);

jobConf.set("totalNumberOfPages", "" + k \* k)

NLineInputFormat.addInputPath(job, new Path(args[1] + "/Job" + (iterationNum - 1)))

jobConf.setInt("mapreduce.input.lineinputformat.linespermap", k \* k / 20)

FileOutputFormat.setOutputPath(job, new Path(args[1] + "/Job" + iterationNum))

waitForCompletion

iterationNum++ **//Increment counter to complete 9 iterations**

}

//Last job to add delta values of last iteration via @LastJobMapper

double currDelta = getGlobalCounter(DELTA\_SUM)

conf = getConf()

job = Job.getInstance(conf, "PageRank MR: " + iterationNum)

job.setJarByClass(PageRankFromSyntheticData.class)

jobConf = job.getConfiguration()

jobConf.set("mapreduce.output.textoutputformat.separator", " ")

job.setMapperClass(LastJobMapper.class)

job.setNumReduceTasks(0); // No reduce, map only job

job.setInputFormatClass(NLineInputFormat.class)

job.setOutputKeyClass(Text.class)

job.setOutputValueClass(PageNode.class)

//Setting global accessible data for job

jobConf.set("delta", "" + currDelta) **//set delta as receiveed from previous job**

jobConf.set("totalNumberOfPages", "" + k \* k) **//set totalPages to k\*k**

NLineInputFormat.addInputPath(job, new Path(args[1] + "/Job" + (iterationNum - 1)))

jobConf.setInt("mapreduce.input.lineinputformat.linespermap", k \* k / 20)

FileOutputFormat.setOutputPath(job, new Path(args[1] + "/Job" + "LastJobOutput"))

waitForCompletion()

System.out.println("The sum of all pages pageRank is: " + (double) getGlobalCounter(PAGE\_RANK\_SUM)/ (10000000)); //output total pagerank sum

}

**How you solved the dangling-page problem ?**

The dangling page problem was solved using the dummy node as described in the course module.

1. The job starts with initializing the delta I.e dummy node pagerank to 0.
2. The map phase reads this delta as zero and in the reduce phase while calculating pagerank for each page, check if page is dummy node.
3. The gobal counter value is set to this value and it is made available to the next job via global counters.
4. The next map phase on receiving this value distributes it over all nodes.
5. An extra map only job is created at the end to add all delta values for the last iteration.

**Running 20 Map Tasks:**

job.setInputFormatClass(NLineInputFormat.class);

NLineInputFormat.addInputPath(job, new Path(args[1] + "/Job" + (iterationNum – 1)));

jobConf.setInt("mapreduce.input.lineinputformat.linespermap", k \* k / 20);

The 20 Map task are created using the the above code. It is evident from the logs and the final map task output that 20 output files are created.

***RUN RESULTS (FOR MR)***

*Program for k=1000 and 10 iterations on the following two configurations(- Machine (m4.large)):*

***5 cheap machines*** *(1 master and 4 workers)*

Running time: 14 Mins

ClusterID: j-2U2VYU5JUVNH7

***9 cheap machines*** *(1 master and 8 workers)*

Running time: 11Mins

ClusterID: j-10FGYNMXODGJ4

**Syslogs Link:*(****Small Cluster: m4.large (4 +1), Large Cluster m4.large(8+1))*

**Spark Local Run**: *(K=100, 10 iterations)*

<https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/blob/master/logs/SparkLocal_K100/K100_LocalRun.txt>

**MR AWS Small Cluster**: *(K=1000, 10 iterations)*

<https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/tree/master/logs/MR_AWS_SMALL>

**MR AWS Large Cluster**: *(K=1000, 10 iterations)*

<https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/tree/master/logs/MR_AWS_LARGE>

**Output Folder Link:**(Includes output from all iteration but final output in [JobLastJobOutput](https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/tree/master/MROutputAWSLarge/JobLastJobOutput) , For MR folder)

**SPARK Local Run**: *(K=100, 10 iterations)*

<https://github.com/2020-F-CS6240/homework-4-spark-rahulpandeycs/tree/master/localOutputK100>

**AWS Small cluster run**: *(K=1000, 10 iterations)*

<https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/tree/master/MROutputAWSSmall>

**AWS Large cluster run**: *(K=1000, 10 iterations)*

<https://github.com/2020-F-CS6240/homework-4-mapreduce-rahulpandeycs/tree/master/MROutputAWSLarge>