CS6240, MW, Assignment4, Hardik Shah

https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4

PageRank in Spark

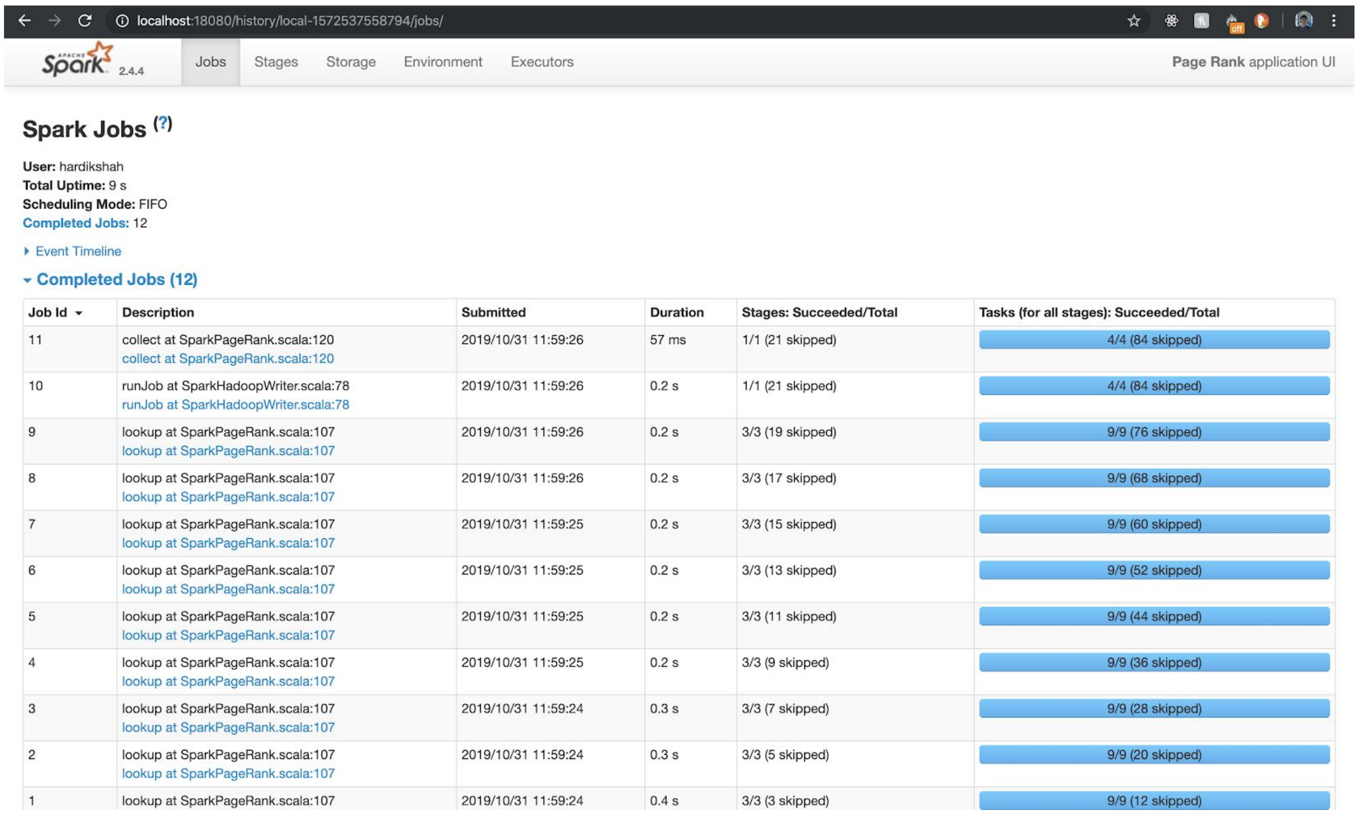
1. Pseudo Code

graphTable is of format (page1, page2) which means page 1 has outlink to page2.
pagerankTable is of format (page, rank) which means page has pagerank ‘rank’.

```
 pageRank(graphTable, pagerankTable) {
       iterations = 10
       k = 100

       // converting table into [rdd(id,[id1, id2...])]
       graphRdd = graphTable.rdd.map(row => (row(0),
                                             List(row(1)).reduceBykey(_ ++ _)
       graphRdd.cache()

       // convert pagerankTable into [rdd(id,pagerank)] and using the same
       // partitioner
       pagerankRdd = pagerankTable.rdd.map(row => (row(0), row(1))
                                  .partitionBy(graphRdd.partitioner)

       for(_ <- 1 to iterations) {

             // joining rdds and calculating outlink contribution, Adding
             //(page, 0.0) to take care of pages with no inlinks
             contrib = graphRdd.join(pagerankRdd).flatmap {
               case(page, (pageslist, rank)) => pageslist.flatMap(
                   outlinkPage =>
                       List((outlinkPage, rank/pageslist.size), (page, 0.0))
                  }.reduceByKey(_+_)

             // find total dangling mass
             delta = contrib.lookup(0).head

             // calculate new page rank
             pagerankRdd = contrib.map {
                   case(edge, value) => if(edge == 0) {
                           (edge,0.0)
                     }
```

```
                      else {
                    (edge, 0.15*(1/k*k) + 0.85 * (value + delta*(1/k*k)))
                    }
                }
            }
            pagerankRdd.saveAsTextFile(outputpath)
  }
```

Note: For all the analysis part, I had enabled saving event logs in the spark-default.confs and ran `./start-history-server.sh` to start the spark history server and navigating to http://localhost:18080 to open Web UI to get details of all the spark jobs.
You can see UI something like this:



Spark Jobs (?)

User: hardikshah
Total Uptime: 9 s
Scheduling Mode: FIFO
Completed Jobs: 12

Event Timeline

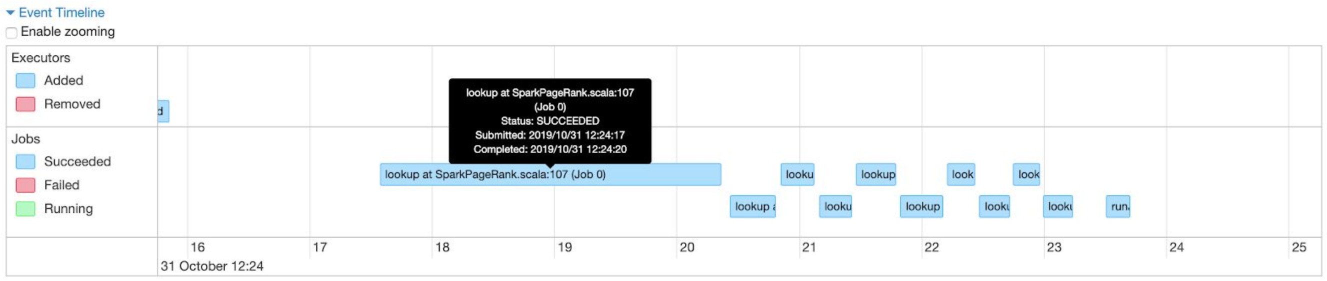
Completed Jobs (12)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
11	collect at SparkPageRank.scala:120 collect at SparkPageRank.scala:120	2019/10/31 11:59:26	57 ms	1/1 (21 skipped)	4/4 (84 skipped)
10	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2019/10/31 11:59:26	0.2 s	1/1 (21 skipped)	4/4 (84 skipped)
9	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:26	0.2 s	3/3 (19 skipped)	9/9 (76 skipped)
8	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:26	0.2 s	3/3 (17 skipped)	9/9 (68 skipped)
7	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:25	0.2 s	3/3 (15 skipped)	9/9 (60 skipped)
6	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:25	0.2 s	3/3 (13 skipped)	9/9 (52 skipped)
5	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:25	0.2 s	3/3 (11 skipped)	9/9 (44 skipped)
4	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:25	0.2 s	3/3 (9 skipped)	9/9 (36 skipped)
3	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:24	0.3 s	3/3 (7 skipped)	9/9 (28 skipped)
2	lookup at SparkPageRank.scala:107 lookup at SparkPageRank.scala:107	2019/10/31 11:59:24	0.3 s	3/3 (5 skipped)	9/9 (20 skipped)
1	lookup at SparkPageRank.scala:107	2019/10/31 11:59:24	0.4 s	3/3 (3 skipped)	9/9 (12 skipped)

2. Operations that perform action

The pseudo code gives an overview of the code for the pagerank algorithm and commands that were used.
According to pseudo code; only commands that triggered action were **lookup()** and **saveAsTextFile()**.

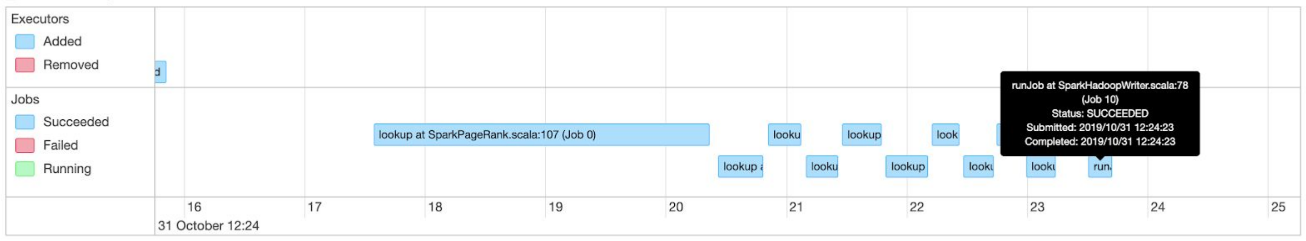
This can be confirmed using the Web UI.
According to the Web UI, the Event Timeline says exactly what's actions were performed by each job and hence, in general the program.



lookup at SparkPageRank.scala:107
(Job 0)
Status: SUCCEEDED
Submitted: 2019/10/31 12:24:17
Completed: 2019/10/31 12:24:20

The above image shows that `lookup()` is an action that triggered the execution of the lineage. There total 10 iterations performed and hence, there are 10 events for `lookup()` since it triggered execution for each job.

The last event in the event timeline was caused by `saveAsTextFile()`.Hence, the last job was caused because of this and thus this is also an action.



runJob at SparkHadoopWriter.scala:78
(Job 10)
Status: SUCCEEDED
Submitted: 2019/10/31 12:24:23
Completed: 2019/10/31 12:24:23

3. Final Page Rank for 10 iterations and k=100

https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/output/output_spark_k100

4. Lineages

Lineage after 1 iteration

```
19/10/29 17:45:34 INFO root: (4) MapPartitionsRDD[19] at map at
SparkPageRank.scala:110 []
 |  ShuffledRDD[18] at reduceByKey at SparkPageRank.scala:106 []
 +-(4) MapPartitionsRDD[17] at flatMap at SparkPageRank.scala:103 []
    |  MapPartitionsRDD[16] at join at SparkPageRank.scala:103 []
    |  MapPartitionsRDD[15] at join at SparkPageRank.scala:103 []
    |  CoGroupedRDD[14] at join at SparkPageRank.scala:103 []
    |  ShuffledRDD[8] at reduceByKey at SparkPageRank.scala:95 []
    |      CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0
B; DiskSize: 0.0 B
    +-(4) MapPartitionsRDD[7] at map at SparkPageRank.scala:94 []
       |  MapPartitionsRDD[6] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[5] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[4] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[1] at createDataFrame at SparkPageRank.scala:49 []
       |  ParallelCollectionRDD[0] at parallelize at SparkPageRank.scala:49 []
    |  ShuffledRDD[13] at partitionBy at SparkPageRank.scala:99 []
    +-(4) MapPartitionsRDD[12] at map at SparkPageRank.scala:98 []
       |  MapPartitionsRDD[11] at rdd at SparkPageRank.scala:98 []
       |  MapPartitionsRDD[10] at rdd at SparkPageRank.scala:98 []
       |  MapPartitionsRDD[9] at rdd at SparkPageRank.scala:98 []
       |  MapPartitionsRDD[3] at createDataFrame at SparkPageRank.scala:55 []
       |  ParallelCollectionRDD[2] at parallelize at SparkPageRank.scala:55 []
```

Lineage after 2 iterations

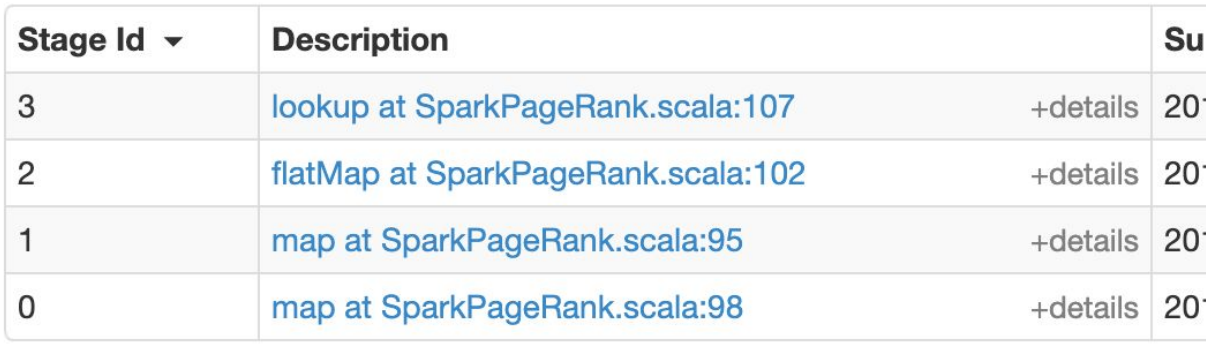
```
19/10/29 17:48:06 INFO root: (4) MapPartitionsRDD[25] at map at
SparkPageRank.scala:110 []
 |  ShuffledRDD[24] at reduceByKey at SparkPageRank.scala:106 []
 +-(4) MapPartitionsRDD[23] at flatMap at SparkPageRank.scala:103 []
    |  MapPartitionsRDD[22] at join at SparkPageRank.scala:103 []
    |  MapPartitionsRDD[21] at join at SparkPageRank.scala:103 []
    |  CoGroupedRDD[20] at join at SparkPageRank.scala:103 []
    |  ShuffledRDD[8] at reduceByKey at SparkPageRank.scala:95 []
    |      CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0
B; DiskSize: 0.0 B
    +-(4) MapPartitionsRDD[7] at map at SparkPageRank.scala:94 []
       |  MapPartitionsRDD[6] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[5] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[4] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[1] at createDataFrame at SparkPageRank.scala:49 []
       |  ParallelCollectionRDD[0] at parallelize at SparkPageRank.scala:49 []
    +-(4) MapPartitionsRDD[19] at map at SparkPageRank.scala:110 []
       |  ShuffledRDD[18] at reduceByKey at SparkPageRank.scala:106 []
       +-(4) MapPartitionsRDD[17] at flatMap at SparkPageRank.scala:103 []
          |  MapPartitionsRDD[16] at join at SparkPageRank.scala:103 []
          |  MapPartitionsRDD[15] at join at SparkPageRank.scala:103 []
          |  CoGroupedRDD[14] at join at SparkPageRank.scala:103 []
          |  ShuffledRDD[8] at reduceByKey at SparkPageRank.scala:95 []
          |      CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize:
0.0 B; DiskSize: 0.0 B
          +-(4) MapPartitionsRDD[7] at map at SparkPageRank.scala:94 []
             |  MapPartitionsRDD[6] at rdd at SparkPageRank.scala:93 []
             |  MapPartitionsRDD[5] at rdd at SparkPageRank.scala:93 []
             |  MapPartitionsRDD[4] at rdd at SparkPageRank.scala:93 []
             |  MapPartitionsRDD[1] at createDataFrame at SparkPageRank.scala:49 []
             |  ParallelCollectionRDD[0] at parallelize at SparkPageRank.scala:49
[]
          |  ShuffledRDD[13] at partitionBy at SparkPageRank.scala:99 []
          +-(4) MapPartitionsRDD[12] at map at SparkPageRank.scala:98 []
             |  MapPartitionsRDD[11] at rdd at SparkPageRank.scala:98 []
             |  MapPartitionsRDD[10] at rdd at SparkPageRank.scala:98 []
             |  MapPartitionsRDD[9] at rdd at SparkPageRank.scala:98 []
             |  MapPartitionsRDD[3] at createDataFrame at SparkPageRank.scala:55 []
             |  ParallelCollectionRDD[2] at parallelize at SparkPageRank.scala:55
[]
```

Lineage after 3 iterations

```
19/10/29 17:49:21 INFO root: (4) MapPartitionsRDD[31] at map at
SparkPageRank.scala:110 []
 |  ShuffledRDD[30] at reduceByKey at SparkPageRank.scala:106 []
 +-(4) MapPartitionsRDD[29] at flatMap at SparkPageRank.scala:103 []
    |  MapPartitionsRDD[28] at join at SparkPageRank.scala:103 []
    |  MapPartitionsRDD[27] at join at SparkPageRank.scala:103 []
    |  CoGroupedRDD[26] at join at SparkPageRank.scala:103 []
    |  ShuffledRDD[8] at reduceByKey at SparkPageRank.scala:95 []
    |      CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize: 0.0
B; DiskSize: 0.0 B
    +-(4) MapPartitionsRDD[7] at map at SparkPageRank.scala:94 []
       |  MapPartitionsRDD[6] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[5] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[4] at rdd at SparkPageRank.scala:93 []
       |  MapPartitionsRDD[1] at createDataFrame at SparkPageRank.scala:49 []
       |  ParallelCollectionRDD[0] at parallelize at SparkPageRank.scala:49 []
    +-(4) MapPartitionsRDD[25] at map at SparkPageRank.scala:110 []
       |  ShuffledRDD[24] at reduceByKey at SparkPageRank.scala:106 []
       +-(4) MapPartitionsRDD[23] at flatMap at SparkPageRank.scala:103 []
          |  MapPartitionsRDD[22] at join at SparkPageRank.scala:103 []
          |  MapPartitionsRDD[21] at join at SparkPageRank.scala:103 []
          |  CoGroupedRDD[20] at join at SparkPageRank.scala:103 []
          |  ShuffledRDD[8] at reduceByKey at SparkPageRank.scala:95 []
          |      CachedPartitions: 4; MemorySize: 820.4 KB; ExternalBlockStoreSize:
0.0 B; DiskSize: 0.0 B
          +-(4) MapPartitionsRDD[7] at map at SparkPageRank.scala:94 []
             |  MapPartitionsRDD[6] at rdd at SparkPageRank.scala:93 []
             |  MapPartitionsRDD[5] at rdd at SparkPageRank.scala:93 []
             |  MapPartitionsRDD[4] at rdd at SparkPageRank.scala:93 []
             |  MapPartitionsRDD[1] at createDataFrame at SparkPageRank.scala:49 []
             |  ParallelCollectionRDD[0] at parallelize at SparkPageRank.scala:49
[]
          +-(4) MapPartitionsRDD[19] at map at SparkPageRank.scala:110 []
             |  ShuffledRDD[18] at reduceByKey at SparkPageRank.scala:106 []
             +-(4) MapPartitionsRDD[17] at flatMap at SparkPageRank.scala:103 []
                |  MapPartitionsRDD[16] at join at SparkPageRank.scala:103 []
                |  MapPartitionsRDD[15] at join at SparkPageRank.scala:103 []
                |  CoGroupedRDD[14] at join at SparkPageRank.scala:103 []
                |  ShuffledRDD[8] at reduceByKey at SparkPageRank.scala:95 []
                |      CachedPartitions: 4; MemorySize: 820.4 KB;
ExternalBlockStoreSize: 0.0 B; DiskSize: 0.0 B
                +-(4) MapPartitionsRDD[7] at map at SparkPageRank.scala:94 []
                   |  MapPartitionsRDD[6] at rdd at SparkPageRank.scala:93 []
                   |  MapPartitionsRDD[5] at rdd at SparkPageRank.scala:93 []
                   |  MapPartitionsRDD[4] at rdd at SparkPageRank.scala:93 []
                   |  MapPartitionsRDD[1] at createDataFrame at
SparkPageRank.scala:49 []
                   |  ParallelCollectionRDD[0] at parallelize at
SparkPageRank.scala:49 []
                |  ShuffledRDD[13] at partitionBy at SparkPageRank.scala:99 []
                +-(4) MapPartitionsRDD[12] at map at SparkPageRank.scala:98 []
                   |  MapPartitionsRDD[11] at rdd at SparkPageRank.scala:98 []
                   |  MapPartitionsRDD[10] at rdd at SparkPageRank.scala:98 []
                   |  MapPartitionsRDD[9] at rdd at SparkPageRank.scala:98 []
                   |  MapPartitionsRDD[3] at createDataFrame at
SparkPageRank.scala:55 []
                   |  ParallelCollectionRDD[2] at parallelize at
SparkPageRank.scala:55 []
```

5. **What was actually executed by a job triggered by your program?**

Job0:

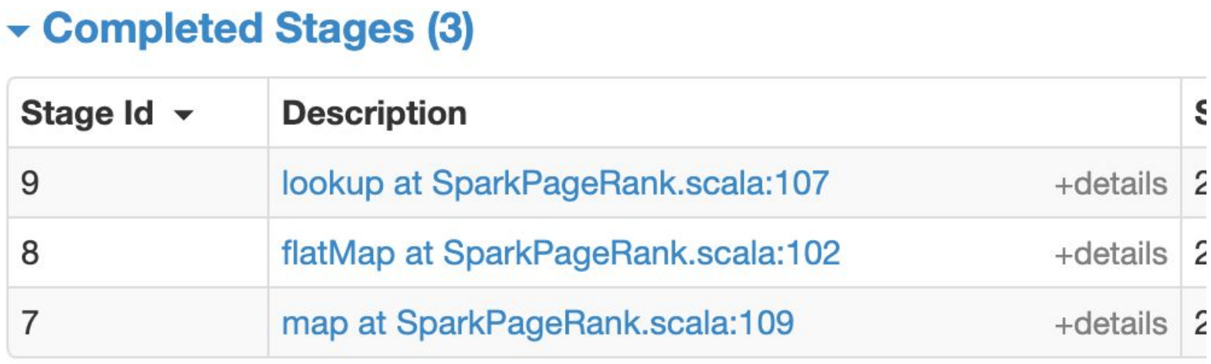


Stage Id	Description		Su
3	lookup at SparkPageRank.scala:107	+details	20
2	flatMap at SparkPageRank.scala:102	+details	20
1	map at SparkPageRank.scala:95	+details	20
0	map at SparkPageRank.scala:98	+details	20

Since it’s the first job, and lookup() is an action that triggered the execution of this job; things that were executed in the job were:

1. creation of pagerankRDD from table (stage 0)
2. creation of graphRDD from table (stage 1)
3. joining of both the rdds and calculating the outgoing contributions (stage 2)
4. lookup() to find the total dangling mass (stage 3).

Job1 - Job9:



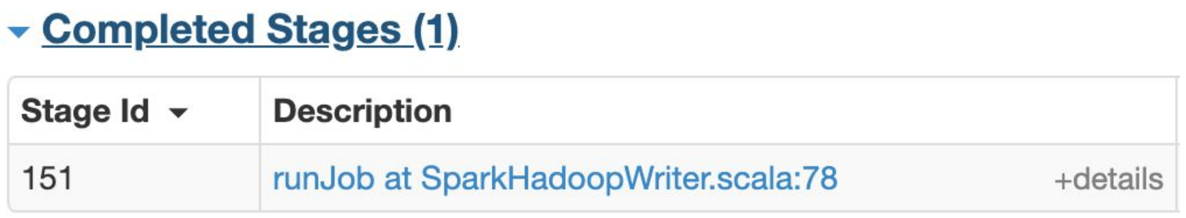
Completed Stages (3)

Stage Id	Description		S
9	lookup at SparkPageRank.scala:107	+details	2
8	flatMap at SparkPageRank.scala:102	+details	2
7	map at SparkPageRank.scala:109	+details	2

All these jobs are part of 10 iterations. Hence, in every Iteration, the things that were executed in particular job were:

1. It computes the pagerank from previous iteration because the join requires pagerank (it was not executed in the previous job because there was no action to trigger it) Stage 7
2. It then computes join and outgoing contributions (stage 8)
3. lookup() for dangling mass which triggered this job (stage 9).

Job 10: The only this executed by this job was `saveToTextFile()`. This is shown below:

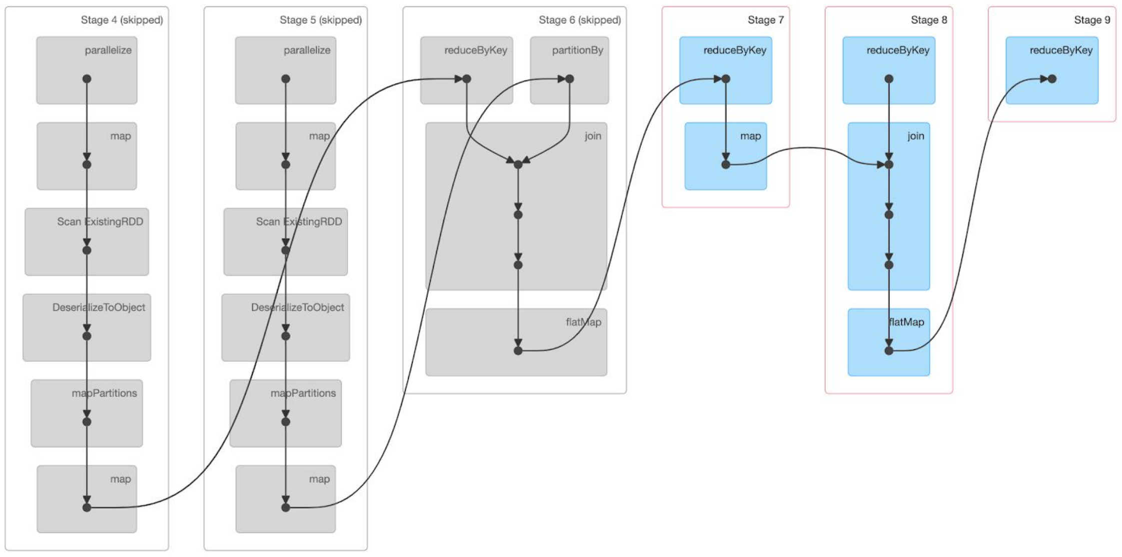


Completed Stages (1)

Stage Id	Description
151	runJob at SparkHadoopWriter.scala:78 +details

6. Is Spark Smart enough to figure out it can reuse rdds computed for earlier action?

Yes. Even if the graph is not cached, Spark is able to use them without recomputing everytime. However, it’s a kind of optimization from the spark side when no shuffling is performed compared to previous iteration. But doesn’t guarantee it would save it to the memory all the time.
If there are more rdd’s coming into the memory that needs to be persisted, spark might remove the previous computed rdd’s from memory.



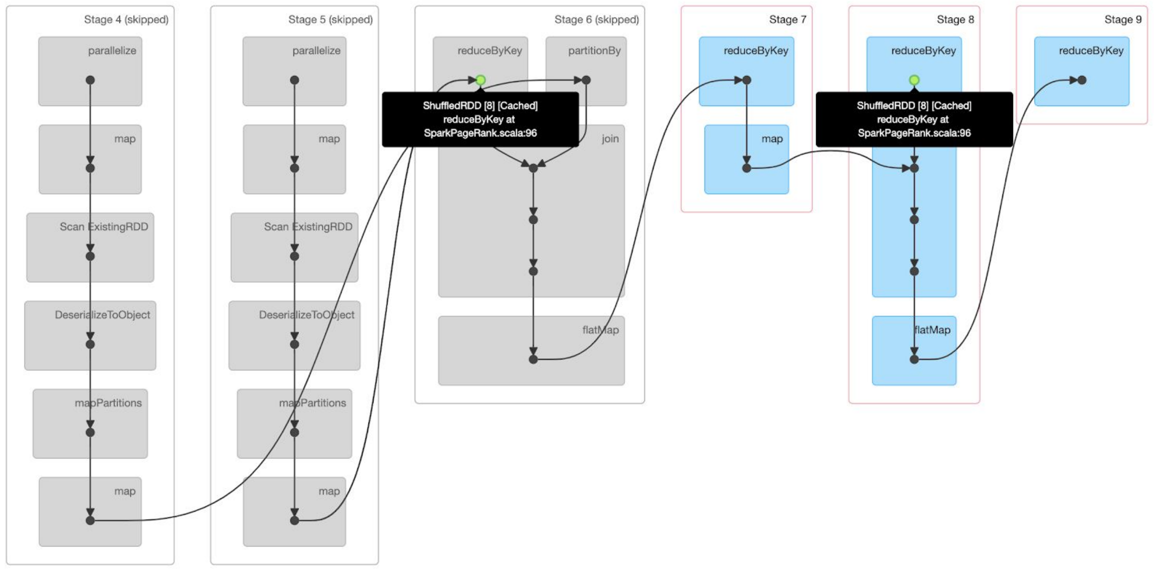
As you can see from the DAG generated from Job2. It shows that all the previous stages were skipped even though Cache was not used (used a separate program without cache).

Also, I put a log in the iterations according to the one of discussions on piazza,

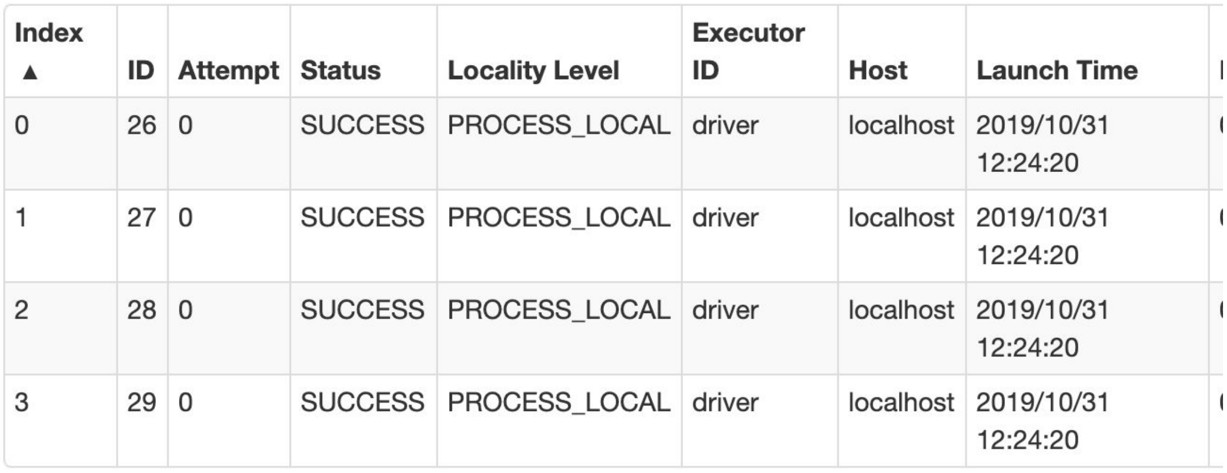
```
for(i <- 1 to iterations) {
      println("iterations:" + i)
      val contri = graph.join(pagerank).flatMap {....}
      val delta = contri.lookup(0).head
      val pageRank = contri.map {...}
}
```

It shows that each iteration was printed just once, instead of reprinting all iterations, which means that spark is reusing the previously computed rdd's.

7. How do persist and cache change this behaviour ?



From DAG, after using cache(), spark uses the rdd's from the cache. The `cache()` or `persist()` computes the Rdd for the first time action is triggered and then it saves it into the cache for reusing it in the next iteration.



Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	I
0	26	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/10/31 12:24:20	(
1	27	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/10/31 12:24:20	(
2	28	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/10/31 12:24:20	(
3	29	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2019/10/31 12:24:20	(

After using cache, the Locality Level for each task changes from ANY to PROCESS_LOCAL which means it uses the rdd's from the memory.
Using `cache()` or `persist()`, you are explicitly telling spark to save the computed rdd's to memory.

PageRank in Map Reduce

1. **Pseudo Code**

   Vertex: represents Vertex Object.
   Vertex has attributes id, pagerank, adjacencyList and contribution. (setters and getters)
   Graph is stored in format “1,[2,3],0.111” => “id,[AdjacencyList],pagerank”

```
 Class PageRankMapper {
           map(Vertex vertex) {
                 id = vertex.getId()
                 adjacencyList = vertex.getAdjacencyList()
                 pageRank = vertex.getPageRank()

                 // get previous iteration's delta from context
                 delta = context.get("delta")
                 totalVertex = context.get("totalVertex")

                 // distribute dangling page mass
                 pageRank = pageRank + 0.85*delta/totalVertex

                 // mass graph from mappers to reducers
                 newVertex = new Vertex(id, adjacencyList, pageRank)

                 emit(id, newVertex)

                 // calculate contribution
                 for each id in adjacencyList:
                   emit(id, pagerank/adjacencyList.size)
                 }
     }

Class PageRankReducer {
      reducer(id, values [vertex, contri1, contri2]) {
            Vertex vertex;
            contriSum = 0.0
            totalVertex = context.get("totalVertex")

            for each value in values:
                  if(value is vertex) {
                        // recover Graph
                        vertex = value
                        }
                        else {
                              //find inlink contribution
                              contriSum += value
                        }

                     newPageRank = 0.15*(1/totalVertex) + 0.85* contriSum
                     vertex.setPageRank(newPageRank)

                     emit(id, vertex)

                     // set delta value so that driver can read for next job.
                    Counter.DELTA.setValue(contriSum)
            }


    Class DeltaMapper {
            map(Vertex vertex) {
                        id = vertex.getId()
                        adjacencyList = vertex.getAdjacencyList()
                        pageRank = vertex.getPageRank()

                        // get previous iteration's delta from context
                        delta = context.get("delta")
                        totalVertex = context.get("totalVertex")

                        // distribute dangling page mass
                        pageRank = pageRank + 0.85* delta/totalVertex

                        // mass graph from mappers to reducers
                       newVertex = new Vertex(id, adjacencyList, pageRank)

                       emit(id, newVertex)
            }
    }

    Driver(args []) {
                iterations = 0
                k = 1000
                delta = 0.0
                inputPath = args(0)
                basePath = args(1)
```

```
        totalVertex = k*k

        // Create first job. Set delta and totalVertex in context.
        Job = createJob(iteration, delta, totalVertex, k)

        Job.setMapper(PageRankMapper)
        Job.setReducer(PageRankReducer)

        // iterate through jobs setting output of previous job to input
        //of next
        // and passing delta of previous job to next
        while(iterations < 10) {
              inputPath = outputPath
              outputPath = basePath + iterations
              delta = Couter.get(DELTA)
              job = createJob(iteration, delta, totalVertex, k)
              Job.setMapper(PageRankMapper)
              Job.setReducer(PageRankReducer)
              iterations ++
        }

       // last job to just add delta values of 10th job.
        inputPath = outputPath
        outputPath = basePath + iterations
        delta = Couter.get(DELTA)
        job = createJob(iteration, delta, totalVertex, k)
        job.setMapper(DeltaMapper)
        job.setNumReducerTasks(0)
}
```

2. **How dangling page problem was solved?**

The dangling page problem was solved using the dummy node. For each page that had no outgoing links, fake edge was created for each of them to the dummy node.
The dummy page had initial page rank of 0. In each iteration, there was dangling mass accumulated on the dangling page, which was distributed to all the pages in the next iteration before next computation.

1. The driver initially sets the delta (dummy page rank) to 0 and sends to the map and reduce phase using context.

2. Map phase computes the contribution initially with delta as 0.
3. In the reduce phase, the page rank is computed for each page. If the page was dummy node, then global counter was set to the delta's value.
4. The driver reads the Global counter value before the execution of next and set's the delta in the context to the counter value.
5. The next map phase calculates the contributions with delta value read from context.
6. There's an extra map only job after the last iteration to add the delta values of the last iteration.

3. **Running Time:**

K, m4.large	5 Cheap Machines(1 and 4)	9 Cheap Machine(1 and 8)
1000	Cluster-id: j-2Y6WJURJ3ZTNP Running Time: 16min	Cluster-id: j-4G6BBS8U3N4C Running Time: 12min

4. **20 Map tasks created**

```
job.setInputFormatClass(NLineInputFormat.class);
NLineInputFormat.addInputPath(job, inputPath);
job.getConfiguration().setInt("mapreduce.input.lineinputformat.linespermap",
k*k/20);
```

NLineInput Format was used to so that at least 20 Map tasks are created.
The logs from the MR program show that each iteration has at least created 20 Map tasks.
And Since, last job was a map only job, you can see 20 output files are generated.

Log Links

Spark k= 100, local:
https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/logs/spark_k100_local

Map Reduce k=1000, small cluster:
https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/logs/mr_k1000_small

Map Reduce k=1000, large cluster:
https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/logs/mr_k1000_large

Output Links

The final page rank output is inside the **final** folder. All the other folders represent the output from the previous iterations.

Spark k= 100, local:
https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/output/output_spark_k100

Map Reduce k=1000, small cluster:
https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/output/output_mr_k1000_small

Map Reduce k=1000, large cluster:
https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-4/tree/master/output/output_mr_k1000_large