CS6240

Rachna Reddy Melacheruvu

HW 4

https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment4/PR-Spark/Spark-Demo

https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment4/PR-MapReduce/MR-Demo

Citation:

1. Changing of jobs – https://coe4bd.github.io/HadoopHowTo/multipleJobsSingle/multipleJobsSingle.html

2. NLineInputFormat - https://stackoverflow.com/questions/28871899/when-to-use-nlineinputformat-in-hadoop-map-reduce
3. To setValue of counter from string to long - https://stackoverflow.com/questions/34403552/hadoop-counters-how-do-i-use-different-type-of-counters
4. To solve the error of intermediate output files not being able to access on s3 - https://github.com/51zero/eel-sdk/issues/356

URLs:

Page Rank Spark-

1. **Project:** **https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment4/PR-Spark/Spark-Demo**
2. **Logs: https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment4/PR-Spark/Spark-Demo/logs**

3. **Output:** **https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment4/PR-Spark/Spark-Demo/output**

Page Rank MapReduce-

1. **Project:** **https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment4/PR-MapReduce/MR-Demo**
2. **Logs :**
    1. **6 machines – https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment4/PR-MapReduce/MR-Demo/1000kfor6/logs**
    2. **11 machines - https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment4/PR-MapReduce/MR-Demo/1000kfor11/Logs/syslog.txt**

3. **Output:**

1. **6 machines – https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment4/PR-MapReduce/MR-Demo/1000kfor6/output/outputFinal/part-r-00003**
2. **11 machines – https://github.ccs.neu.edu/melacheruvur/cs6240/blob/master/Assignment4/PR-MapReduce/MR-Demo/1000kfor11/Output/outputFinal/part-r-00016**

Page Rank in Spark

Pseudo-code

```
val k: Int = args(0).toInt                                  // k as input
val iters: Int = 10                                         // number of iterations to be performed
val maxK: Int = k*k                                         // k^2 required to calculate initial Page Rank
val initialRank: Float = (1/maxK.toFloat)
val vertices = List.range(1,maxK+1)                         // list of vertices from 1 to k
val graph = sc.parallelize(vertices.map(vertex =>{          // creates graph RDD as 1-> 2 and for every k th vertex
                                                            // it points to dummy vertex as k -> 0
 if(vertex%k==0)
  (vertex,0)
 else
  (vertex,vertex+1)
}))
val dummyRDD = List(0).map(x=>(x,0.0f))
val verRanks = vertices.map(vertex=>(vertex,initialRank))    // to calculate ranks for each vertex with initial PR
var ranks = sc.parallelize(verRanks.union(dummyRDD))

for (i <- 1 to iters) {                                     // Page Rank Algorithm Iteration

 val contribs = graph.join(ranks)                           // Join graph and ranks so we get vertex, adjacency list
  .values.map{                                              // and page rank
  case(u,v)=>(u,v)                                          // store only vertex and it's incoming PR
 }
 val combineRanks = contribs.reduceByKey(_ + _)             // add all incoming PR for each vertex
 val dummyRank = combineRanks.lookup(0).head.toFloat        // handling PageRank Mass by distributing it to all vertices
 val PRMass = dummyRank/maxK
 val nextRanks  = ranks.leftOuterJoin(combineRanks).map({   //  to add new PR to each vertex and if vertex has no
  case (k,(u,None)) =>                                      // incoming link, then just add the dangling mass
   {
    if(k==0)
     (k,0.0f)
    else
     (k,PRMass)
   }
  case (k,(u,Some(v))) =>
   {
    if(k==0)
     (k,0.0f)
    else
     (k,v+PRMass)
   }
 })
 ranks = nextRanks                                          // newly computed ranks are given to ranks variable for
```

```
                                                        // next iteration
  var pagerankvalue = ranks.values.sum()                // calculating total PR of the graph
  logger.info("Page Rank Value at-"+i+"th iteration is "+pagerankvalue)
}
logger.info(ranks.toDebugString)
val output = ranks.filter(_._1<=100)                    // writing output to file of 100 vertices
output.saveAsTextFile(args(1))
```

Actions in Program

saveAsTextFile
sum
lookup

Ran program for k=100 and 10 iterations. Output shows PageRanks for vertices 0 to 100. Output in above provided URL

Lineage for RDD Ranks

1. **For 1 iteration**

```
(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
 |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
 |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
 |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
 +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
 +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
    |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
    |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
    |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
    |  CoGroupedRDD[2] at join at PageRank.scala:48 []
    +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
    +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
```

2. **For 2 iterations**

```
(4) MapPartitionsRDD[23] at map at PageRank.scala:59 []
 |  MapPartitionsRDD[22] at leftOuterJoin at PageRank.scala:59 []
 |  MapPartitionsRDD[21] at leftOuterJoin at PageRank.scala:59 []
 |  CoGroupedRDD[20] at leftOuterJoin at PageRank.scala:59 []
 +-(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
 |  |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
 |  |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
 |  |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
 |  +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |  |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
 |  +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
 |     |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
 |     |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
 |     |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
 |     |  CoGroupedRDD[2] at join at PageRank.scala:48 []
 |     +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
 |     +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |  ShuffledRDD[19] at reduceByKey at PageRank.scala:54 []
 +-(4) MapPartitionsRDD[18] at map at PageRank.scala:49 []
    |  MapPartitionsRDD[17] at values at PageRank.scala:49 []
```

```
  |  MapPartitionsRDD[16] at join at PageRank.scala:48 []
  |  MapPartitionsRDD[15] at join at PageRank.scala:48 []
  |  CoGroupedRDD[14] at join at PageRank.scala:48 []
  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
  +-(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
     |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
     |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
     |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
     +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
     |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
     +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
        |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
        |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
        |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
        |  CoGroupedRDD[2] at join at PageRank.scala:48 []
        +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
        +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
```

3. For 3 iterations

```
(4) MapPartitionsRDD[35] at map at PageRank.scala:59 []
 |  MapPartitionsRDD[34] at leftOuterJoin at PageRank.scala:59 []
 |  MapPartitionsRDD[33] at leftOuterJoin at PageRank.scala:59 []
 |  CoGroupedRDD[32] at leftOuterJoin at PageRank.scala:59 []
 +-(4) MapPartitionsRDD[23] at map at PageRank.scala:59 []
 |  |  MapPartitionsRDD[22] at leftOuterJoin at PageRank.scala:59 []
 |  |  MapPartitionsRDD[21] at leftOuterJoin at PageRank.scala:59 []
 |  |  CoGroupedRDD[20] at leftOuterJoin at PageRank.scala:59 []
 |  +-(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
 |  |  |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
 |  |  |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
 |  |  |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
 |  |  +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |  |  |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
 |  |  +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
 |  |     |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
 |  |     |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
 |  |     |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
 |  |     |  CoGroupedRDD[2] at join at PageRank.scala:48 []
 |  |     +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
 |  |     +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |  |  ShuffledRDD[19] at reduceByKey at PageRank.scala:54 []
 |  +-(4) MapPartitionsRDD[18] at map at PageRank.scala:49 []
 |     |  MapPartitionsRDD[17] at values at PageRank.scala:49 []
 |     |  MapPartitionsRDD[16] at join at PageRank.scala:48 []
 |     |  MapPartitionsRDD[15] at join at PageRank.scala:48 []
 |     |  CoGroupedRDD[14] at join at PageRank.scala:48 []
 |     +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
 |     +-(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
 |        |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
 |        |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
 |        |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
 |        +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |        |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
 |        +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
 |           |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
 |           |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
 |           |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
 |           |  CoGroupedRDD[2] at join at PageRank.scala:48 []
 |           +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
 |           +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
 |  ShuffledRDD[31] at reduceByKey at PageRank.scala:54 []
```

```
+-(4) MapPartitionsRDD[30] at map at PageRank.scala:49 []
  |  MapPartitionsRDD[29] at values at PageRank.scala:49 []
  |  MapPartitionsRDD[28] at join at PageRank.scala:48 []
  |  MapPartitionsRDD[27] at join at PageRank.scala:48 []
  |  CoGroupedRDD[26] at join at PageRank.scala:48 []
  +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
  +-(4) MapPartitionsRDD[23] at map at PageRank.scala:59 []
    |  MapPartitionsRDD[22] at leftOuterJoin at PageRank.scala:59 []
    |  MapPartitionsRDD[21] at leftOuterJoin at PageRank.scala:59 []
    |  CoGroupedRDD[20] at leftOuterJoin at PageRank.scala:59 []
    +-(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
    |  |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
    |  |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
    |  |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
    |  +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
    |  |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
    |  +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
    |     |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
    |     |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
    |     |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
    |     |  CoGroupedRDD[2] at join at PageRank.scala:48 []
    |     +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
    |     +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
    |  ShuffledRDD[19] at reduceByKey at PageRank.scala:54 []
    +-(4) MapPartitionsRDD[18] at map at PageRank.scala:49 []
      |  MapPartitionsRDD[17] at values at PageRank.scala:49 []
      |  MapPartitionsRDD[16] at join at PageRank.scala:48 []
      |  MapPartitionsRDD[15] at join at PageRank.scala:48 []
      |  CoGroupedRDD[14] at join at PageRank.scala:48 []
      +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
      +-(4) MapPartitionsRDD[11] at map at PageRank.scala:59 []
        |  MapPartitionsRDD[10] at leftOuterJoin at PageRank.scala:59 []
        |  MapPartitionsRDD[9] at leftOuterJoin at PageRank.scala:59 []
        |  CoGroupedRDD[8] at leftOuterJoin at PageRank.scala:59 []
        +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
        |  ShuffledRDD[7] at reduceByKey at PageRank.scala:54 []
        +-(4) MapPartitionsRDD[6] at map at PageRank.scala:49 []
          |  MapPartitionsRDD[5] at values at PageRank.scala:49 []
          |  MapPartitionsRDD[4] at join at PageRank.scala:48 []
          |  MapPartitionsRDD[3] at join at PageRank.scala:48 []
          |  CoGroupedRDD[2] at join at PageRank.scala:48 []
          +-(4) ParallelCollectionRDD[0] at parallelize at PageRank.scala:33 []
          +-(4) ParallelCollectionRDD[1] at parallelize at PageRank.scala:42 []
```

What was executed by a job triggered by the program

The program makes use of toDebugString to see the lineage of the "ranks" RDD. Once transformation leftOuterJoin is used on ranks RDD, that's when it is required to compute the RDD as there are actions to be performed on the RDD. This job triggers the execution of the ranks RDD.

Observations

1. **Is Spark smart enough to figure out that it can re-use RDDs computed for an earlier action?**
   No. RDD just contains a reference to the task that has to be performed in the lineage. RDD does not store any data.

RDD operations are lazy. When an action is called, then through the lineage, each required RDD is computed and then used. Until action is called, RDDs are just tracked in lineage and are not computed. These RDDs are not stored in memory if not told by cache() or persist(). If cache() or persist() are not used, RDDs are not stored. For every action or iteration that requires a particular RDD, it's always computed again and again by executing the lineage. Since RDDs don't store data in them, re-use of RDDs are not possible.

2. **How do persist() and cache() change this behavior?**
   persist() and cache() are methods in Spark to save the computed RDDs in memory.
   These above two methods are lazy as well. Only when an action is used on RDD, then its computed and then if cache() or persist() are used, it's stored in memory.
   By storing in memory, if the same computed RDD is required for any other action or in the next iteration, it can be fetched from memory and can avoid re-computation. Re-use of RDD is possible in this case. If memory is full with a lot of data from RDD, then spark automatically switches to re-computation of RDD, rather than storing it.
   Difference between cache() and persist() is that cache() can store in MEMORY_ONLY. Whereas, with persist(), various storage levels can be chosen from.
   When there is RDD partitioning required or re-use of RDD in iterations, it's better to use cache() or persist() to store it in memory.

Page Rank in Spark

Pseudo-code

To generate a graph (Map Only) –

```
Mapper {
        map(k)
        {
            for( int i=1 to k*k)                    // create a graph from 1 to k^2
            {
                if(i%k==0)                          // for every kth vertex outgoing link doesn't exists
                    emit(i , 0)
                else
                    emit(i, i+1)                    // all other vertices have outgoing link to next vertex
             }

        }
}
```

To generate adjacency list with initial page rank

Input in form – vertex1, vertex2
vertex1, vertex3

Output in form – vertex1 vertex2#vertex3;page_rank

```
Mapper {
map()
{
    String s = covert value to string
    String[] followers = s.split(",")                   // split string on delimited as ","
```

```
    write a key-value pair as (followers[0],followers[1]) to context
}
}
Reducer {
reduce(key,values)
{
    String adj = "";
    for each value of values for the key
    {
        adj += node + "#";                                  // every adjacent vertex is separated by #
    }
    String outputVal = adj+";"+(1/k*k)                      // value is adjacency list and initial pagerank separated by ";"
    Context.write(key,outputVal)
   }
}
```

To Calculate Page Rank for n iterations(2 jobs)

1st Job to calculate page rank

2nd Job to add dangling page mass to each vertex's PR

```
Counter PR_Mass{
        dangling_mass                                       //counter to keep track of dangling mass
}
```

JOB 1

```
Mapper {
        map((id n, vertex N)
        {
        emit(n, N)                                          // Pass along the graph structure
        p = N.pageRank / N.adjacencyList.size()             // Compute contributions to send  along outgoing
        links
        for all m in N.adjacencyList do
              emit( m, p )
        }
}

Reducer {
        reduce((id m, [p1, p2,…])
        {
              If(m!=0)
              {
                  s = 0
                  M = NULL
                  for all p in [p1, p2,…] do
                     if isVertex(p) then                    // The vertex object was found: recover graph structure
                          M = p
                     Else
                        s += p                              // A PageRank contribution from an inlink was found:
                                                            // add it to the running sum.
                     M.pageRank = (1-α)/|V| + α*s           // PR formula
                  emit( m, M)
               }
```

```
            Else                                        // if it's vertex 0 then add the dangling mass and
            {                                           // update counter
                s =0
                for all p in [p1, p2,…] do
                    s+=p
                PR_mass.dangling_mass = s /|V|
            }
        }
}
```

JOB 2

```
Mapper {
        map((id n, vertex N)
        {
            emit(n, N)                                  // Pass along the graph structure
        }
}

Reducer {
        dMass = PR_mass.dangling_mass                   // dangling mass to be added to each vertex's PR
        reduce((id m, [p]])
        {
            String[ ] adj = p.split(";")                // split value by ";" to get the PR
            Float newPR = adj[1].toInt()+dMass          // add dMass to old PR
            String outputVal = adj[0]+";"+newPR
            emit(m,outputVal)
        }
}
```

To add Page Ranks of every vertex

```
        Mapper
        {
          map((id n, vertex N)
          {
            String[] inputStr = s.split("\t")           // split string on delimited as "\t"
            String[] PRStr = inputStr.split(";")        // split adjaceny list and PR
            String emitPR = PRStr[1]                    // consider page rank only
            emit("PR", emitPR)                          // emit same key for all and value as PR
          }
        }

        Reducer
        {
            reduce("PR",[p1,p2,p3….])
            {
                finalPR =0
                For p in all values do
                    finalPR + = p
                emit("final PR", finalPR)
            }
        }
```

How was dangling-page problem handled?

Dummy vertex is add with key 0 for every vertex which doesn't have an outgoing link. In the reducer job, all page rank values of 0 vertices is aggregated and divided by the number of vertices in the graph. That result called the dangling mass is updated to every vertex's page rank value.
As shown above in the pseudo code, this is done with 2 jobs.
1 job to only calculate the page rank for every vertex ignoring the dangling mass.
2^{nd} job to add the dangling mass to every vertex.
This dangling mass is updated in the global counter in the reducer of the 1^{st} job. Dangling page mass is updated to the configuration object in the driver program. And the updated dangling mass is used in reducer of the 2^{nd} job where, for each vertex, (dangling mass * α) is added to it's page rank.
This way the dangling mass is distributed to each vertex.

Usage of NLineInputFormat

Used NLineInputFormat to pass 50000 lines as input for each Mapper.
As there will be 1000,000 line in the file which holds the Adjacency list and page rank, each Mapper receives 50000 lines of input.
With this, for every job which uses NLineInputFormat class:
Number of Map tasks = 22
Number of Reduce tasks = 12
NLineInputFormat class is used in the following files of the code: PRCalc.java, PageRank.java, PageRankAgg.java .

Running Time Measurements

Map-Reduce:

For k = 1000

	6 machines	***11 machines***
Start Time	00:49:05	01:40: 38
End Time	01:13:42	01:59:11
Time Taken (in sec)	24 min 37 sec	18 min 33 sec
Page Rank of 1000 pages	0.9946458287277518	0.9946458287277516