CS 6240 Parallel Data Processing in MapReduce - 02
Homework 4
Dheeraj Joshi

Q. Describe the steps taken by Spark to execute your source code. In particular, for each method invocation of your Scala Spark program, give a brief high-level description of how Spark processes the data. (10 points)

Steps taken by Spark for computing PageRank for all pages:

1. Initialize Spark Configuration and Context for processing the input data. Also, initialize default values to be used for PageRank computation e.g. alpha, main graph RDD, empty set, etc.

2. Spark context reads the input text file line by line and applies the parsing map function using the java parser used in the previous assignment. Filter this newly formed RDD to remove pages with invalid outlink sets and characters.

3. Using this newly formed RDD, burst the values(sets) into a new RDD consisting of all outlinks as keys and form a union with the main RDD. This results in a perfect graph structure now consisting of the "Sink Nodes" as well.

4. Using the above main graph RDD, find the page count. Using this page count, we initialize the tuple consisting of PageRank i.e. (1/ page count) and the set of outlinks for every page.

5. Iterating 10 times do the following:

(I) Aggregate the PageRank for all pages with no outlinks to get the delta value for the new PageRank computation.

(II) Subtract the set of dangling nodes from the main graph RDD and assign new PageRank Values to 0.0.

(III) For all pages with outlinks, form an RDD with (Key, Value) pairs consisting of the link and its PageRank contribution. (PageRank / Size of Outlinks Set)

(IV) Union of (II) & (III) RDD's results in a consolidated RDD with new PageRank contributions for all pages.

(V) Join (IV) with main graph RDD and compute the new PageRank using the new contributions and (I). Finally, form a tuple for the main graph as the new PageRank and the outlink sets.

6. Sorting the main graph RDD by PageRank in the descending order, we take the top 100 and form a new RDD by applying coalesce to get a single file.

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 4
Dheeraj Joshi

Q. Compare the Hadoop MapReduce and Spark implementations of PageRank. (10 points)

Part 1: For each line of your Scala Spark program, describe where and how the respective functionality is implemented in your Hadoop jobs.

Spark Steps 1 to 4:

```
val input = sc.textFile(args(0)).map(line => line.split(":")(0) ->
Bz2WikiParser.getLinkPages(line))
  .filter(x => x._2 != null)

//Conversion from Java Set to Scala Set
val inputRDD = input.map(x => x._1 -> x._2.asScala).persist

//Burst the values of the above Graph into a list of nodes and combine with the above
graph
//This adds the sink nodes to the list
val resultRDD = inputRDD
  .flatMap(_._2.map(link => (link,emptySet)))
  .union(inputRDD).reduceByKey((x,y) => x ++ y)

//Get total page count from the consolidated RDD
var pageCount = resultRDD.count

//Map every graph tuple to Key,Value Pair as (PageName -> (Initial PageRank,Set of
Outlinks)) in the Main Graph
inputPageRankGraphRDD = resultRDD.map(x => x._1 -> ((1.0/pageCount), x._2)).persist
```

For Spark steps 1 to 4, the corresponding Hadoop steps were as follows:

A separate parsing job that would construct the graph structure and get the total page count for PR computation. Emit every page with its corresponding set in the map phase. Also, for accumulating sink nodes, emit every outlink page with an empty set. In the reduce phase, aggregating values by key gave us the main graph structure for PR computation.

Spark Step 5:

```
//Delta Value Aggregator
//We aggregate pagerank of all dangling nodes to form the delta factor for PR
computation from the Main Graph
val deltaValue = inputPageRankGraphRDD.filter(x => x._2._2.isEmpty).aggregate(0.0)(
  (acc,x) => (acc + x._2._1),
  (acc1,acc2) => (acc1+ acc2))

//RDD for accumulating all inlink contributions from the consolidated set of outlinks
//We reduce by key to accumulate the constributions
//Each contribution is the (PR of the page/size of the outlinks set)
val inlinkContributingRDD = inputPageRankGraphRDD.filter(x =>
x._2._2.nonEmpty).values.flatMap{
  case(x,y) =>
    y.map(link => link -> x/y.size)
}.reduceByKey(_+_)
```

```
//RDD for separating the dangling nodes that don't have any outlinks
val danglingNodeRDD = inputPageRankGraphRDD.subtractByKey(inlinkContributingRDD)

//TransformedRDD where we initialize the current page Rank of the dangling node to 0
for further computation
val transformedDanglingRDD = danglingNodeRDD.map(x => x._1 -> (0.0))

//Union of the RDD with Inlink contributions & Transformed Dangling Node RDD
val allNewContributionsRDD = inlinkContributingRDD.union(transformedDanglingRDD)

//Joined RDD for Union of Total Consolidated Contributions & Dangling Nodes with the
Main Graph
//New PR is computed and set to the tuple along with it's outlink set (PR,Set)
inputPageRankGraphRDD = inputPageRankGraphRDD.join(allNewContributionsRDD)
  .map(x => x._1 -> ((alpha/pageCount) + (1-alpha)*(x._2._2+
deltaValue/pageCount),x._2._1._2)).persist
```

For Spark Step 5, the corresponding Hadoop steps were as follows:

The output of the parser was read again as input by the PR computation job. For the 1st Iteration, the PageRank is set to initial PR emitted to the reduce phase in similar fashion as the parsing job. The reduce phase then simply accumulates contributions by key and computes the new PR. Although for 10 Iterations, the delta takes an extra iteration to be accumulated in the context variable as map reduce does not allow global accumulators across two phases.

Spark Step 6:

```
//Array for sorting by PageRank field in the descending order from every Tuple in the
Main Graph and retrieve top 100
val topK = inputPageRankGraphRDD.sortBy(_._2._1,false).map(x => x._1 ->
x._2._1).take(100)

//Form an RDD of the above array by using Spark Context and performing coalesce on the
RDD pairs
val topKRDD = sc.parallelize(topK,1).coalesce(1)
```

For Spark Step 6, the corresponding Hadoop Steps were as follows:

The output of the PR Computation Job in the 11th Iteration is used as input for the Top K Job. The map job simply emitted local top 100 pages based on PR and the reducer then emitted the global 100 as the output.

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 4
Dheeraj Joshi

Part 2: Discuss the advantages and shortcomings of the different approaches. This could include, but is not limited to, expressiveness and flexibility of API, applicability to PageRank, available optimizations, memory and disk data footprint, and source code verbosity.

Advantages/Shortcomings of Spark and Hadoop MapReduce Approaches:

- Apache Spark processes all data in-memory while Hadoop MapReduce persists back to the disk after a map or reduce action, so spark does perform a few operations faster than the map reduce approach.
- From general design, the spark approach is well suited for iterative computations that need to update the same data again and again. Map reduce is fit for one-pass jobs such as parsing or transformation. One could argue that spark performs best when all data can fit in memory, whereas map reduce performs well for data that needs to be transferred to disk and retrieved.

- Spark provides a parallel data structure called RDD's for various direct transformation and actions to be performed. Although, these operations are costlier resource wise when performed across several machines. The machines need higher physical memory for processing data.

- Spark has comfortable API's for several languages whereas Hadoop Map reduce is restricted to Java and uses other tools for better extensibility.

- Spark has better source code verbosity because of a functional programming structure which allow in place transformations and actions on RDD's. Map Reduce strictly follows object oriented design

Q. Report for both configurations the Spark execution time. For comparison, also include the total execution time (from pre-processing to top-k) of the corresponding Hadoop executions from Assignment 3. (4 points)

6 Machines (1 Master 5 Slaves)			
Approach	**Start Time**	**End Time**	**Total Time**
Spark	6:28:07	8:30:45	2:02:38
MapReduce	19:36:52	20:45:17	1:08:25

11 Machines (1 Master 10 Slaves)			
Approach	**Start Time**	**End Time**	**Total Time**
Spark	5:11:19	6:10:39	0:59:20
MapReduce	0:49:15	1:28:32	0:39:17

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 4
Dheeraj Joshi

Q. Discuss which system is faster and briefly explain what could be the main reason for this performance difference. (4 points)

The Map Reduce approach works faster in this particular case due to the following reasons:

- Spark uses yarn as the master and due to it's container allocation strategy, at any point in execution new containers cannot be added until the existing phase has been completed. As the map reduce approach stores data into disk, the adding of containers isn't a bottleneck.

- Spark provides in place transformations but this results in higher amounts of shuffling data across the nodes. Noting that both approaches use serialized objects to read and write, the number of machines and types of machines used were similar for both approaches. Because of it's in memory design, the Spark approach clearly would have performed well if a machine type with higher physical memory was allocated. As a proof, the AWS website states specs for a m4.large system used in both approaches as follows:

Model	vCPU	Mem	SSD Storage (GB)	Dedicated EBS Bandwidth (Mbps)
m4.large	2	8	EBS-only	450

- From even further observations, we see that the page rank iterative computations are performed faster than in the map reduce approach by spark.

Q. Report the top-100 Wikipedia pages with the highest PageRank, along with their PageRank values, sorted from highest to lowest, for both the simple and full datasets, from both the Spark and MapReduce execution. Are the results the same? If not, try to find possible explanations. (4 points)

From the results of Top-100 for both approaches, we can clearly admit that the results are same. Although, there are a few pages interchanged in order, but we could argue that is due to loss of lower precision bits. Also, the application of the delta value from the 1st iteration itself might result in difference of a few page ranks in the spark approach than the map reduce approach.