CS 6240 Assignment 4 Priyanka Shenoy

Theoretical data read and referenced from official Spark website (http://spark.apache.org/docs/latest/programming-guide.html)

Question 1 -

Steps taken by Spark to execute code -

- Spark runs sets of processes on a cluster, coordinated by SparkContext the main program i.e. DriverProgram.
- On a specific cluster, the SparkContext can connect to types of cluster managers (standalone or YARN), which allocate resources across applications.
- Once connected,executors are run by Spark on nodes, which are processes that run computations and store data.
- Sparks sends application code (JAR file) to the executors.
- SparkContext sends tasks to the executors to run.

Function	Description - Data
map	Return a new RDD by applying a function to all elements of this RDD.
filter	Return a new RDD containing only the elements that satisfy a predicate
flatMap	Return a new RDD by by flattening the current RDD
keyBy	Creates tuples of the elements in this RDD by applying function
join	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
reduceByKey	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V
subtractByKey	Return an RDD with the pairs from current set whose keys are not in other.
union	Return a new dataset that contains the union of the elements in the source dataset and the

	argument.
mapValues	Return a new RDD by applying a function to only values of this RDD (map performs on key as well as values)
takeOrdered	Return the first *n* elements of the RDD using either their natural order or a custom comparator
reverse	Reverses the string column and returns it as a new string column

Question 2-

Map Reduce VS Scala implementation

MapReduce	Scala

PageRankPreprocessing.java
 	
Map - Gets data from the BzParser. Creates a custom object with node and outlinks. Outlinks is empty for dangling nodes
Reduce - Converts outlinks into string for processing
	val inputValues = sparkCont.textFile(args(0), sparkCont.defaultParallelism)
	.map(line => PageRankPreProcessing.*readXML*(line))
	.filter(line => !line.contains(**"IncorrectValue"**))
	.map(line => line.split(**"prishen"**))
	.map(line => **if** (line.length == 1) {
	(line(0), *List*()) }
	else {
	(line(0), line(1).split(**"~"**).toList)
	})

PageRankPreprocessing.java Map - Gets data from the BzParser. Creates a custom object with node and outlinks. Outlinks is empty for dangling nodes Reduce - Converts outlinks into string for processing	**var** uniqueNodeWithLinks = inputValues.values .flatMap { node => node } .keyBy(node => node) .map(line => (line._1, *List*[String]())).union(inputValues).reduceByKey((value1, value2) => value1.++(value2))
DriverProgram.java	**var** uniqueNodeWithPageRank =

PageRankJob.java

10 iterations are done at PageRankJob

Map - Gets data from PageRankPreprocessing and assigns initial page ranks to all nodes. Also emits dummy in case of dangling nodes.Map updates pageranks by adding the delta values for iteration i in next iteration

Reduce - Calculates page ranks according to new dangling nodes and assigns to all available nodes.

```
uniqueNodeWithLinks.keys
 .map(line => (line, initialPageRank))

for (i <- 1 to loop) {
 try {
  var danglingValue =
sparkCont.accumulator(0.0)
  var pageRankSetValues =
uniqueNodeWithLinks.join(uniqueNodeWithP
ageRank)
    .values
    .flatMap {
     case (links, pageRank) =>
      val size = links.size
      if (size == 0) {
        danglingValue += pageRank
        List()                                    }
else {
        links.map(url => (url, pageRank /
size))
       }
    }.reduceByKey(_ + _)


uniqueNodeWithPageRank.subtractByKey(pa
geRankSetValues)
 .map(rec => (rec._1
,0.0)).union(pageRankSetValues)
 .mapValues
 [Double](pageRankAcc => alpha *
initialPageRank +
  (1 - alpha) * (danglingValueAcc /
noOfNodes + pageRankAcc))
```

PageRankTopK.java

Map- Gets data from PageRankJob (and PageRankDeltaJob)
And filters values of top 100 pages with highest ranks locally (for each map)

Reduce - Calculates global top 100 pages for given pages

```
var sortedVal =
uniqueNodeWithPageRank.takeOrdered(100)
(Ordering[Double]
 .reverse.on { line => line._2 })

sparkCont.parallelize(sortedVal)
 .saveAsTextFile(args(1))
```

Comparison Factors	Hadoop	Spark
Memory and disk data footprint	Hadoop kills job as it completes. It reads and writes data from the disk. Memory used is very little but causes high data footprint	Performs better if data fits into available spark memory. Since there is less file writing, disk footprint is less
Source Code Verbosity	Long and verbose code	Short and concise
Fault Tolerance	Relies on disk for data. There are multiple copies and backups of the same. In case of failure, job can continue from where it crashed.	Processing needs to start from beginning in case of crash
Expressiveness and flexibility of API	No interactive command line hence slightly cumbersome programming. Though it has more API's for assistance	Spark is easier to program with interactive mode.It is has API's for java,scala and python
Applicability to PageRank	Both can be used. In theory Spark is supposed to be better. But other above factors determine the final outcome	Both can be used. In theory Spark is supposed to be better. But other above factors determine the final outcome
Cost	Disk cheaper than memory hence ends up being less expensive	Memory more expensive than disk hence ends up being expensive

Question 3-

Execution time of Scala program on EMR

Node	Spark (seconds)	Hadoop (seconds)
5 Nodes	5890	3822
10 Nodes	3202	2221

Output files for both scala and java executions are present in output folder. Top 100 pagerank names remain the same but pagerank values may slightly differ due to precision on different systems.

According to all the theory that I have studied in module and read, Scala is supposed to take lesser execution time than normal MR in Java. Reason for the same is in memory calculation. In my case it does not so happen.

One of the reason could be, due to insufficient proficiency in Scala language I am not able to completely understand and persist RDD the way they are meant to be. Partitioning,reduce and map methods maybe used inefficiently. This might be slowing the process.

Additionally, a large part of the data load and processing is done in the pre-processing java file. If, Spark infrastructure does not by default run Java program parallely, this would lead to a bottleneck. At this stage I can say, I tried my best to implement code as efficiently as possible with my current abilities. There is scope for improvement.