Sravani Beeram

CS6240 Parallel Data Processing using Mapreduce
Homework 4

Design Discussion

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.

Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in main program (*driver program*).In regard to a specific cluster,SparkContext can connect to several types of cluster managers such as standalone or Yarn which allocates resources across applications.After connection,spark runs executors on nodes in the cluster which are processes that run computations and store data for the application Next,it sends application code i.e JAR files to the executors and finally SparkContext sends tasks to the executors to run.

Method	Description
filter	Returns a new RDD containing only the elements that satisfy a predicate
flatMap	Returns a new RDD by flattening the results
map	Returns a new RDD by applying a function to all elements of current RDD
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
keyBy	Creates tuples of the elements in the RDD by applying function
subtractByKey	Returns an RDD with the pairs current RDD whose keys are not in other RDD
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

union	Returns a new dataset that contains the union of the elements in the source dataset and the argument
mapValues	Returns a new RDD by applying a function to only values of this RDD (map performs on key as well as values)
takeOrdered	Returns 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

Compare the Hadoop MapReduce and Spark implementations of PageRank.

Scala	Hadoop

`var nodesList = sc.textFile(args(0), sc.defaultParallelism)`
`    .map(line => Bz2WikiParser.parse(line))`
`    .filter(line => !line.contains("Invalid"))`
`    .map(line => line.split("SB"))`
`    .map(line => if(line.size == 1)`
`    {`
`     (line(0),List())        }`
`    else{`
`     (line(0),line(1).split("~").toList)`
`    })`
 	
`var nodesWithDangling = nodesList.values`
`        .flatMap{node => node}`
`        .keyBy(node => node)`
`        .map(line => (line._1 ,List[String]()))`
`val finalNodes = nodesList.union(nodesWithDangling)`
`      .reduceByKey((value1,value2) =>`
`value1.++(value2))`
	preProcessing.java contains the corresponding program in Hadoop
	
	Mapper : Calls the Bz2WikiParser to parse the input file and emits a custom data structure with pagename and its link pages. It emits pageName ,adjacency list
	
	Reducer : Appends all the linkPages and sets a default page rank

`var nodeListWithPageRank = finalNodes.keys`
`        .map(line => (line,initial_pageRank))`
 	
`for(i <- 1 to iterationCount){`
`  var danglingFactor =  sc.accumulator(0.0)`
`  var pageRankData =`
	pageRankCal.java contains corresponding code in Hadoop
	
	pageRankJob.java calls pageRankCal.java for 10 iterations

```
finalNodes.join(nodeListWithPageRank).values
                         .flatMap{
                          case(outLinks, pageRank) => {
                            val size = outLinks.size
                           if(size == 0){
                             danglingFactor +=  pageRank
                             List()
                           }else{
                             outLinks.map(url =>
(url,pageRank/size))

                           }
                          }
                         }
                         .reduceByKey(_+_)
  pageRankData.first
  val danglingValue : Double = danglingFactor.value

  var missingData  =
nodeListWithPageRank.subtractByKey(pageRankData)
  var allNodes = missingData.map(page  => (page._1,0.0))
                      .union(pageRankData)
  nodeListWithPageRank =
allNodes.mapValues[Double](accumulatedPageRank =>
(alpha * initial_pageRank + (1 - alpha) * (danglingValue /
totalPages + accumulatedPageRank)))
}
```
	Mapper: Gets data from preProcessing.java , assigns initial page rank. Emits dummy and page rank for dangling nodes.For link pages it emits page name and its page rank.It updates dangling nodes page rank using delta value
	
	Reducer:updates the delta value and calculates page rank for nodes with adjaceny list

```
**var temp = nodeListWithPageRank.takeOrdered(100)
              (*Ordering*[Double].reverse.on{line => line._2})

sc.parallelize(temp)
 .saveAsTextFile(args(1))**
```
	TopK.java contains corresponding job.
	
	Mapper: Receives input from pageRankCal.java.It computes the local top 100
	
	Reducer: Computes global top 100

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.

Properties	Spark	Hadoop
Memory and Disk data footprint	Performs better when all the data fits in the memory, especially on dedicated clusters	Hadoop is designed for data that doesn't fit in the memory.It kills its processes as soon as a job is done ,so it can run well alongside other services
Expressiveness and flexibility of API	Spark is easier to program and includes an interactive mode.It is comfortable for API's for java,scala and python	Difficult to program but has many tools to make it easier
Source Code verbosity	Less code verbosity	High code verbosity
Applicability to pagerank	Spark and Hadoop both can be used for pagerank.Best solution can be decided based on the differences mentioned in first three points	Spark and Hadoop both can be used for pagerank.Best solution can be decided based on the differences mentioned in first three points
Cost	Spark is more cost-effective since less hardware can perform the same tasks much faster	I could be cheaper because of available tools
Failure Tolerance	If a process crashes in the middle of execution, it will have to start processing from the beginning	It relies on hard drives, so if a process crashes in the middle, it could continue where it less off

Performance Comparison

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

	Hadoop(min)	Spark(min)
Run 1 (1 Master 5 Workers)	63.97	99.5
Run 2 (1 Master 10 Workers)	41.1	55.2

Discuss which system is faster and briefly explain what could be the main reason for this performance difference

According to my understanding Scala should take lesser time than Mapreduce to execute . Mapreduce takes a long time to write things to disk and read them back. Elimination of this restriction makes Spark faster.But, it's not the same case above. Main problem could be because of the code design.Inefficient use of the available methods may result in such differences.It may also be because of Bz2WikiParser.java program which does the pre-processing.

Report the top-100 Wikipedia pages with the highest PageRanks, 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

Outputs for hadoop,spark for simple, full datasets are in the folder named “output files for report”. Yes, all the results of all runs are same.