Turnitin Originality Report

Processed on: 30-Apr-2020 8:14 PM CDT
ID: 1311796695
Word Count: 804
Submitted: 2

BigDataFinalProject By Kiran Noolvi

Similarity Index	Similarity by Source	
7%	Internet Sources:	5%
	Publications:	2%
	Student Papers:	7%

2% match (student papers from 29-Nov-2019)
Submitted to UT, Dallas on 2019-11-29

2% match (student papers from 29-Apr-2020)
Submitted to University of Missouri, Kansas City on 2020-04-29

1% match (Internet from 27-Apr-2020)
https://dzone.com/articles/apache-spark-performance-tuning-straggler-tasks

1% match (student papers from 23-Sep-2016)
Submitted to City University on 2016-09-23

1% match (student papers from 02-Oct-2017)
Submitted to Birkbeck College on 2017-10-02

package bigdataproject import org.apache.log4j.{Level, Logger} import org.apache.spark.graphx.{Edge, Graph, VertexId} import org.apache.spark. util.LongAccumulator import org.apache.spark.{ SparkConf, SparkContext} import scala.collection.mutable import scala.collection.mutable.{ArrayBuffer, ListBuffer} import scala.util.control.Breaks object PopularSuperHeroCalculation { var startingSuperHeroID:Long = 5306 var targetSuperHeroID:Long = 6306 var targetBoolean: Boolean = false var targetReached:Option[LongAccumulator] = None //Explanation: Connections, Distance, Status of whether the node has been processed type nodeInfo = (Array[Long], String, Long) //Each node of the graph we will be constructing, Character ID with Node Info type heroNode = (Long, nodeInfo) //We should be mapping the hero id with their respective hero names def constructNamedDict(line: String) : Option[(VertexId, String)] = { val lineData = line.split('\"') if(lineData.length<=1){ //return none if we do not have sufficient information in a line None }else{ val marvelCharId = lineData(0).trim().toLong val marvelCharName = lineData(1).toString if(marvelCharId<=6486){ //idNameDict+=(marvelCharId->marvelCharName.toString) Some (marvelCharId, marvelCharName) //return hero id mapped with the hero name }else{ None } } } //Construct graph edges def constructHeroEdges(line: String) : List[Edge[Int]] = { var connectionEdge = new ListBuffer[Edge[Int]]() val eachLine = line.split("") for (i <- 1 to (eachLine.length - 1)) { connectionEdge += Edge(eachLine(0).toLong, eachLine(i).toLong, 0) } connectionEdge.toList } def constructGraph(line:String): heroNode = { val eachLine = line.split("\\s+") //Extract the hero id val marvelCharId = eachLine(0).toLong //We will mark the status of each node as NOTTOUCHED var visitedStatus:String = "NOTTOUCHED" var degreeSeparation: Long= 999999999 val heroNeighbors:

ArrayBuffer[Long] = ArrayBuffer() for(i <- 1 to eachLine.length-1){ heroNeighbors.append(eachLine(i).toLong) } if(marvelCharId == startingSuperHeroID){ visitedStatus = "TOUCHED" degreeSeparation = 0 } (marvelCharId,(heroNeighbors.toArray,visitedStatus,degreeSeparation)) } def superHeroConnection(line:String):(Long, Int) ={ val eachLine = line.split("\\s") val superHeroId = eachLine(0).toLong val noOfConnections = eachLine.length-1 (superHeroId, noOfConnections) } def bfsAlgorithm(marvelNode:heroNode):Array[heroNode]={ val marvelCharId:Long = marvelNode._1 //Get the connections of each super hero val heroNeighbors:Array[Long] = marvelNode._2._1 //Get the status of the node var visitedStatus:String = marvelNode._2._2 //Get the distance of separation val degreeSeparation:Long = marvelNode._2._3 val graph:ArrayBuffer[heroNode] = ArrayBuffer() if(visitedStatus=="TOUCHED"){ for(neighbor <- heroNeighbors){ val newMarvelCharId = neighbor val newDegreeSeparation = degreeSeparation + 1 val newVisitedStatus = "TOUCHED" if(targetSuperHeroID==neighbor){ targetBoolean=true if(targetReached.isDefined){ targetReached.get.add(1) } } val newHeroNode:heroNode = (newMarvelCharId,(Array(), newVisitedStatus,newDegreeSeparation)) graph.append(newHeroNode) } visitedStatus = "PROCESSED" } graph.append((marvelCharId, (heroNeighbors,visitedStatus,degreeSeparation))) graph.toArray } def reduceGraph(superHero1:nodeInfo, superHero2:nodeInfo):nodeInfo = { val hero1Neighbors:Array[Long] = superHero1._1 val hero1VisitedStatus:String = superHero1._2 val hero1DegreeSep:Long = superHero1._3 val hero2Neighbors:Array[Long] = superHero2._1 val hero2VisitedStatus:String = superHero2._2 val hero2DegreeSep:Long = superHero2._3 var degreeSeparation:Long = 999999999 var visitedStatus:String = "NOTTOUCHED" var heroNeighbors:ArrayBuffer[Long] = ArrayBuffer() if(hero1Neighbors.length>=1){ heroNeighbors ++= hero1Neighbors } if(hero2Neighbors.length>=1){ heroNeighbors ++= hero2Neighbors } if(hero1DegreeSepdef main(args:Array[String]): Unit = { //Just log errors if any Logger.getLogger(" org ").setLevel(Level.ERROR) val sc = new SparkContext(new SparkConf().setAppName("Popular Super Hero")) var graphPath:String ="" var namesPath:String ="" var resultsOutputPath:String ="" if(args.length<3){ println("Invalid command to execute: Formal should have Marvel Graph dataset path, Marvel Names dataset path, Path where results have to be written") System.exit(1) }else{ if(args.length== 5){ startingSuperHeroID = args(3).toLong targetSuperHeroID = args(4).toLong } graphPath = args(0) namesPath = args(1) resultsOutputPath = args(2) } val marvelGraphData = sc.textFile(graphPath) val marvelNameData = sc.textFile(namesPath) val idNameMap = marvelNameData.flatMap(constructNamedDict) val marvelGraphEdges = marvelGraphData.flatMap(constructHeroEdges) val graph = Graph(idNameMap, marvelGraphEdges, "Nobody") val graphJoined = graph.degrees.join(idNameMap) val top10Famous = graphJoined.sortBy(_._2._1, ascending=false).take(5) var count = 0; val resultString = new mutable.StringBuilder() val newDictForSH = sc.textFile(namesPath).filter(x=>x.split('\"').length>1).map(x=> (x.split('\"')(0).trim().toLo ng,x.split('\"')(1))) val idNameDict = newDictForSH.collect.toMap //println("****"+myMap.getOrElse(14, "UNk")) resultString.append("\nRESULTS:\n") resultString.append("\nTop 5 famous characters calculated using graphX are: \n") for(famousSuperHero<-top10Famous){ count += 1 resultString.append("Top "+count+" famous hero: \n") resultString.append("\tSuper Hero ID: " + famousSuperHero._1+"\n") resultString.append("\tSuper Hero Name: "+famousSuperHero._2._2+"\n") //resultString.append("\tSuper Hero Connections: "+famousSuperHero._2._1+"\n") } val superHeroConnections = marvelGraphData.map(superHeroConnection)

```
resultString.append("\n*******************************************\n")
resultString.append("\n*******************************************\n")
resultString.append("\nTop 5 famous characters calculated without using
graphX are: \n") val top5withoutGraphX = superHeroConnections
.reduceByKey((x,y)=>x+y).map(x=>(x._2, x._1)).sortByKey(false).take(
5) count = 0 for(row <-top5withoutGraphX){ count += 1
resultString.append("Top "+count+" famous hero: \n")
resultString.append("\tSuper Hero ID: " + row._2+"\n") val
superHeroName = idNameDict.get(row._2.toLong).getOrElse("Unknown")
resultString.append("\tSuper Hero Name: "+superHeroName+"\n")
//resultString.append("\tSuper Hero Connections: "+row._1+"\n") }
resultString.append("\n*******************************************\n")
resultString.append("\n*******************************************\n")
var marvelGraphRDD = marvelGraphData.map(constructGraph)
targetReached = Some(sc.longAccumulator("targetReached")) val forLoop
= new Breaks var degreeSeparation: Int = 0;
resultString.append("\nStarting BFS from the super hero:
"+idNameDict.get(startingSuperHeroID).getOrElse("Unknown")+"\n") val
timeStart = System.currentTimeMillis() forLoop.breakable{ for(iteration <-
1 to 20){ degreeSeparation = iteration val afterBFS =
marvelGraphRDD.flatMap(bfsAlgorithm) println("Processing "+
afterBFS.count()+" values") if((targetReached.isDefined &&
targetReached.get.value > 0) || targetBoolean==true){
resultString.append("We have reached the target super hero:
"+idNameDict.get(targetSuperHeroID).getOrElse("Unknown"))
forLoop.break } marvelGraphRDD = afterBFS.reduceByKey(reduceGraph)
} } val timeEnd = System.currentTimeMillis() resultString.append("\nTime
taken to reach from the super hero
'"+idNameDict.get(startingSuperHeroID).getOrElse("Unknown")+"' to
'"+idNameDict.get(targetSuperHeroID).getOrElse("Unknown")+"' is "+
(timeEnd- timeStart)+" milliseconds\n") resultString.append("\nDistance
of separation between the Super Hero's '"+
idNameDict.get(startingSuperHeroID).getOrElse("Unknown") +"' and
'"+idNameDict.get(targetSuperHeroID).getOrElse("Unknown")+"' is " +
degreeSeparation+"\n") //Display results on the console
println(resultString) //Writing output to a file
sc.parallelize(Seq(resultString.toString())).saveAsTextFile(resultsOutputPath)
println("Finished running successfully") } }
```