databricks Demo of Page Rank Calculation 2023-10-30 16:15:32

(https://databricks.com)

```
import org.apache.spark.HashPartitioner

val links  = sc.parallelize(List(("MapR", List("A","B")), ))
```

```
import org.apache.spark.HashPartitioner
```

```
import org.apache.spark.HashPartitioner

val links = sc.parallelize(List(("MapR",List("Baidu","Blogger")),("Baidu", List("MapR")),("Blogger",List("Google","Baidu")),("Google",
()
var ranks = links.mapValues(v => 1.0)
```

```
import org.apache.spark.HashPartitioner
links: org.apache.spark.rdd.RDD[(String, List[String])] = ShuffledRDD[1] at partitionBy at command-1660944887008766:3
ranks: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[2] at mapValues at command-1660944887008766:4
```

```
// val ranks = contributions.reduceByKey((x,y) => x + y).mapValues(v => 0.15+0.85*v)
```

```
val contributions = links.join(ranks).flatMap { case (url, (links, rank)) => links.map(dest => (dest, rank / links.size)) }
```

```
contributions: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[6] at flatMap at command-1660944887008768:1
```

```
val ranks = contributions.reduceByKey((x, y) => x + y).mapValues(v => 0.15 + 0.85*v)
```

```
ranks: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[8] at mapValues at command-1660944887008769:1
```

```
ranks.collect
```

```
res1: Array[(String, Double)] = Array((Google,0.575), (MapR,1.8499999999999999), (Blogger,0.575), (Baidu,1.0))
```

```
val lines= spark.read.textFile("dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/links.txt").rdd
val iters = 20
val links = lines.map{ s =>
      val parts = s.split("\\s+")
      (parts(0), parts(1))
    }.distinct().groupByKey().cache()

 var ranks = links.mapValues(v => 1.0)

    for (i <- 1 to iters) {
      val contribs = links.join(ranks).values.flatMap{ case (urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank / size))
      }
      ranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)
    }
    val output = ranks.collect()
    output.foreach(tup => println(tup._1 + " has rank:" + tup._2))
    println("==================")
    output.foreach(tup => println(tup._1 + " has rank:" + f"${tup._2}%.3f"))
    println("==================")
ranks.collect()
val r = ranks.toDF("URL", "PageRank")
r.show()
```

```
a has rank:1.0
b has rank:1.0455994483347224
c has rank:1.038759531084514
===================
a has rank:1.000
b has rank:1.046
c has rank:1.039
===================
+---+------------------+
|URL|          PageRank|
+---+------------------+
|  a|               1.0|
|  b|1.0455994483347224|
|  c| 1.038759531084514|
+---+------------------+

lines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[883] at rdd at command-1660944887008771:1
iters: Int = 20
links: org.apache.spark.rdd.RDD[(String, Iterable[String])] = ShuffledRDD[888] at groupByKey at command-1660944887008771:6
ranks: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[1029] at mapValues at command-1660944887008771:15
output: Array[(String, Double)] = Array((a,1.0), (b,1.0455994483347224), (c,1.038759531084514))
```

```
import org.apache.spark.sql.SparkSession
r.createOrReplaceTempView("Table_2")
val r1=sqlContext.sql("select PageRank from Table_2 where PageRank <2")
```

```
import org.apache.spark.sql.SparkSession
r1: org.apache.spark.sql.DataFrame = [PageRank: double]

formattedArray: org.apache.spark.sql.Dataset[String] = [value: string]
res16: Array[String] = Array(1.000, 1.046, 1.039)
```