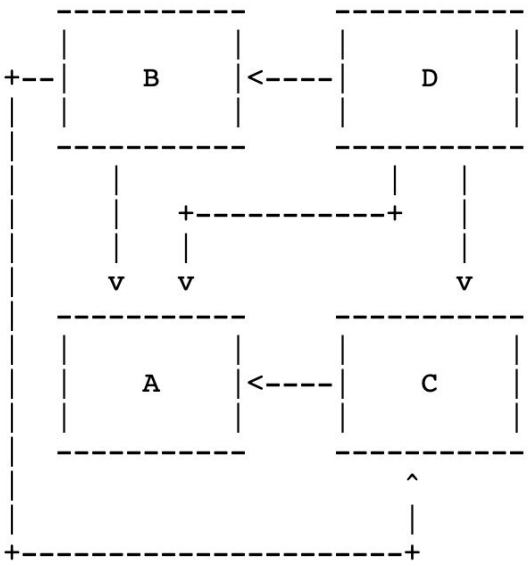
PageRank on GCP

Emily Weng CS570 Big Data

Pagerank

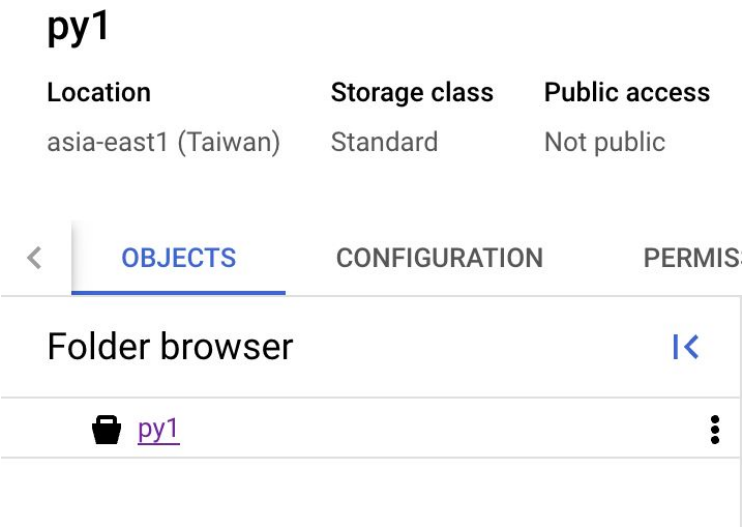
What is pagerank?

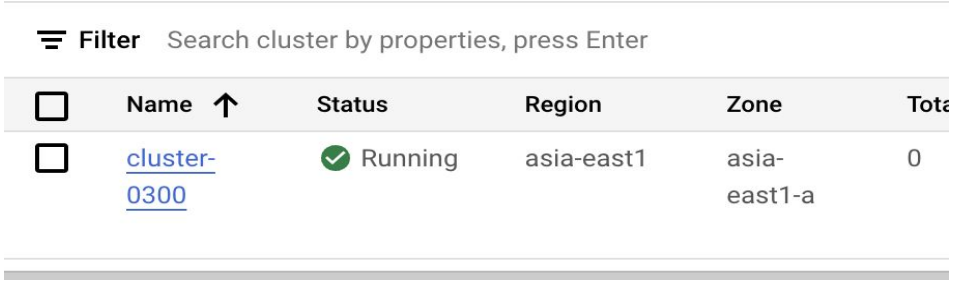
1. This determines the ranking of a page.
2. How would we know? By which page is most “popular”
3. If each page is 1
4. Then:
    a. A's PageRank is: PR(A) = (1-d) + d * (PR(B) / 2 + PR(C) / 1 + PR(D) / 3)
    b. B's PageRank is: PR(B) = (1-d) + d * (PR(D) / 3)
    c. C's PageRank is: PR(C) = (1-d) + d * (PR(B) / 2 + PR(D) / 3)
    d. D's PageRank is: PR(D) = 1-d
    e. Damping factor is 0.85



Set up Pyspark on GCP

Create bucket and cluster





Connecting to the Master Node using ssh

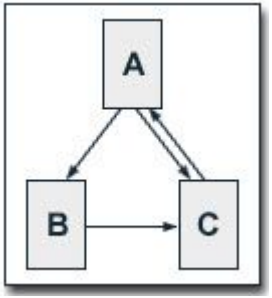
```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
eweng909@cluster-0300-m:~$
```

Do this question using PySpark

Pagerank

Assuming

- the initial PageRank value for each webpage is 1.
- the damping factor is 0.85
- the relation of the webpages is:



Calculations

1. Manually calculate the first 2 iteration of the PageRank
2. First iteration:
    a. A = 1
    b. B = (1/2) = 0.5
    c. C = 1 + (1/2) = 1.5
    d. PageRank (A) = 1 – 0.85 + 0.85 * 1 = 1
    e. PageRank (B) = 1 – 0.85 + 0.85 * 1 = 0.575
    f. PageRank (C) = 1 – 0.85 + 0.85 * 1.5 = 1.425

Calculations

1. Second iteration:
    a. A = 1
    b. B = (1/2) = 0.5
    c. C = 0.575 + (1/2) = 1.075
    d. PageRank (A) = 1 – 0.85 + 0.85 * 1.425 = 1.36125
    e. PageRank (B) = 1 – 0.85 + 0.85 * 0.5 = 0.575
    PageRank (C) = 1 – 0.85 + 0.85 * 1.075 = 1.06375

Prepare Data in HDFS

SSH-in-browser

```
A B
A C
B C
C A
~
```

1. Manual input data
   a. vi pagerank_data.txt
2. Create a directory (folder) to store the data
   a. hdfs dfs -mkdir hdfs:///mydata
   b. hdfs dfs -put pagerank_data.txt hdfs:///mydata
3. To verify that the file is indeed located in the mydata folder, run the following command:
   a. hdfs dfs -ls hdfs:///mydata

```
eweng909@cluster-0300-m:~$ vi pagerank_data.txt
eweng909@cluster-0300-m:~$ hdfs dfs -mkdir hdfs:///mydata
eweng909@cluster-0300-m:~$ hdfs dfs -put pagerank_data.txt hdfs:///mydata
eweng909@cluster-0300-m:~$ hdfs dfs -ls hdfs:///mydata
Found 1 items
-rw-r--r--   1 eweng909 hadoop         16 2024-06-25 14:27 hdfs:///mydata/pagerank_data.txt
eweng909@cluster-0300-m:~$
```

Prepare the program

1. Create a py file:
   a. vi pagerank.py
2. Can get more at github

```
import re
import sys
from operator import add
from pyspark.sql import SparkSession
def computeContribs(urls, rank):
    """Calculates URL contributions to the rank of other URLs."""
    num_urls = len(urls)
    for url in urls:
        yield (url, rank / num_urls)


def parseNeighbors(urls):
    """Parses a urls pair string into urls pair."""
    parts = re.split(r'\s+', urls)
    return parts[0], parts[1]


if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: pagerank <file> <iterations>", file=sys.stderr)
        sys.exit(-1)

    print("WARN: This is a naive implementation of PageRank and is given as an example!\n" +
          "Please refer to PageRank implementation provided by graphx",
          file=sys.stderr)

    # Initialize the spark context.
    spark = SparkSession\
        .builder\
        .appName("PythonPageRank")\
        .getOrCreate()
```

Running the program with Pyspark

1. spark-submit pagerank.py hdfs:///mydata/pagerank_data.txt 1

```
A has rank: 1.0.
C has rank: 1.4249999999999998.
B has rank: 0.575.
eweng909@cluster-0300-m:~$
```

Running the program with Pyspark

1. spark-submit pagerank.py hdfs:///mydata/pagerank_data.txt 2

```
C has rank: 1.06375.
A has rank: 1.3612499999999996.
B has rank: 0.575.
eweng909@cluster-0300-m:~$
```

Running the program with Pyspark

1. spark-submit pagerank.py hdfs:///mydata/pagerank_data.txt 10

```
A has rank: 1.1667391764027368.
C has rank: 1.1900114118087488.
B has rank: 0.6432494117885129.
eweng909@cluster-0300-m:~$
```

PageRank + Scala + GCP

Set up Scala on GCP

1. install scala
    a. curl -fL https://github.com/coursier/launchers/raw/master/cs-x86_64-pc-linux.gz | gzip -d > cs && chmod +x cs && ./cs setup
        i. Curl wasn't working for me so instead I uploaded the file onto my bucket using this download link:
        ii. https://downloads.lightbend.com/scala/2.13.10/scala-2.13.10.tgz
    b. Check if you see the folder scala-2.13.10:
        i. gsutil ls gs://py1/scala-2.13.10
    c. Download files from bucket to cluster:
        i. gsutil cp -r gs://py1/scala-2.13.10 ~/

Set up Scala on GCP

1. Verify download:
   a.

```
eweng909@cluster-0300-m:~$ ls ~/scala-2.13.10
LICENSE  NOTICE  bin  doc  lib  man
eweng909@cluster-0300-m:~$
```

Do this question using Scala

1. Prepare data
    a. Manually input data:
        i. pagerank_data.txt

Create a directory to store the data

1. Create a directory (folder) to store the data
   a. hdfs dfs -mkdir hdfs:///mydata
   b. hdfs dfs -put pagerank_data.txt hdfs:///mydata
2. To verify that the file is indeed located in the mydata folder, run the following command:
   a. hdfs dfs -ls hdfs:///mydata

```
eweng909@cluster-0300-m:~$ hdfs dfs -ls hdfs:///mydata
Found 1 items
-rw-r--r--   1 eweng909 hadoop         16 2024-06-25 14:27 hdfs:///mydata/pagerank_data.txt
eweng909@cluster-0300-m:~$
```

Run the program and see results

1. Execute
    a. spark-shell
    b. You will be brought into spark with scala
2. Input code and get results:
3. Change this line to get different runs:
    a. for (i <- 1 to 10)

Program:

```
val lines = sc.textFile("hdfs:///mydata/pagerank_data.txt")

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 10) {
    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 + "."))

ctx.stop()
~
```

Results:

```
scala> val lines = sc.textFile("hdfs:///mydata/pagerank_data.txt")
lines: org.apache.spark.rdd.RDD[String] = hdfs:///mydata/pagerank_data.txt MapPartitionsRDD[1] at textFile at <
console>:23

scala> val links = lines.map{ s =>
     |     val parts = s.split("\\s+")
     |     (parts(0), parts(1))
     | }.distinct().groupByKey().cache()
links: org.apache.spark.rdd.RDD[(String, Iterable[String])] = ShuffledRDD[6] at groupByKey at <console>:26

scala> var ranks = links.mapValues(v => 1.0)
ranks: org.apache.spark.rdd.RDD[(String, Double)] = MapPartitionsRDD[7] at mapValues at <console>:23

scala> for (i <- 1 to 10) {
     |     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 * _)
     |
     | }

scala> val output = ranks.collect()
output: Array[(String, Double)] = Array((B,0.6432494117885129), (A,1.1667391764027368), (C,1.1900114118087488))

scala> output.foreach(tup => println(tup._1 + " has rank: " + tup._2 + "."))
B has rank: 0.6432494117885129.
A has rank: 1.1667391764027368.
C has rank: 1.1900114118087488.
```

Close cluster and bucket and you're done!