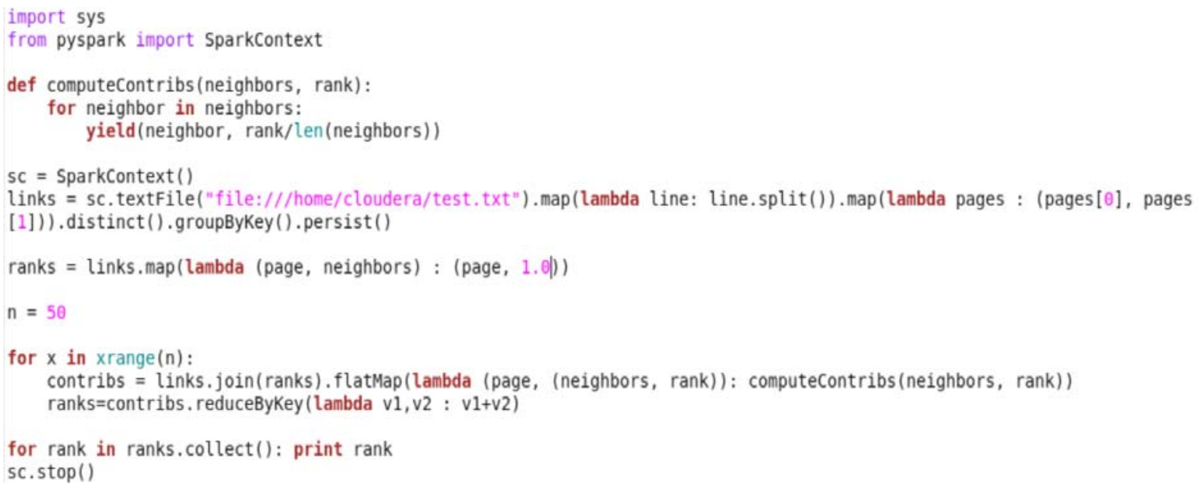
Adrien Xie, 451852 Ziqian Luo, 466739 xie.z

Problem 1

(a) PageRank without teleportation

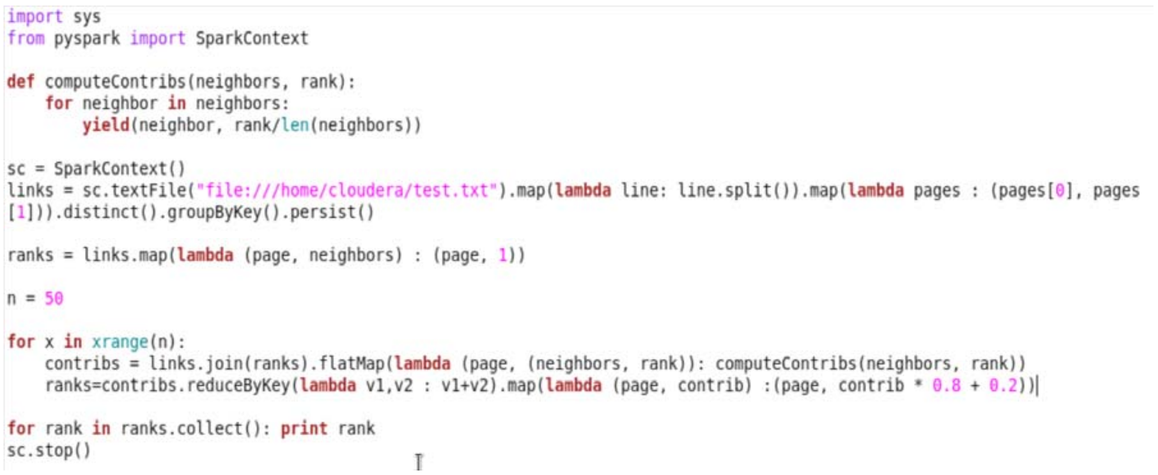
- Results:
  (u'a', 0.69230769230769251)
  (u'b', 0.92307692307692335)
  (u'c', 1.384615384615385)
- Spark code :



```
import sys
from pyspark import SparkContext

def computeContribs(neighbors, rank):
    for neighbor in neighbors:
        yield(neighbor, rank/len(neighbors))

sc = SparkContext()
links = sc.textFile("file:///home/cloudera/test.txt").map(lambda line: line.split()).map(lambda pages : (pages[0], pages
[1])).distinct().groupByKey().persist()

ranks = links.map(lambda (page, neighbors) : (page, 1.0))

n = 50

for x in xrange(n):
    contribs = links.join(ranks).flatMap(lambda (page, (neighbors, rank)): computeContribs(neighbors, rank))
    ranks=contribs.reduceByKey(lambda v1,v2 : v1+v2)

for rank in ranks.collect(): print rank
sc.stop()
```

(b) PageRank with teleportation

- Results:
  (u'a', 0.77776789683221303)
  (u'b', 0.92591275133183948)
  (u'c', 1.2962765344051668)
- Spark code:



```
import sys
from pyspark import SparkContext

def computeContribs(neighbors, rank):
    for neighbor in neighbors:
        yield(neighbor, rank/len(neighbors))

sc = SparkContext()
links = sc.textFile("file:///home/cloudera/test.txt").map(lambda line: line.split()).map(lambda pages : (pages[0], pages
[1])).distinct().groupByKey().persist()

ranks = links.map(lambda (page, neighbors) : (page, 1))

n = 50

for x in xrange(n):
    contribs = links.join(ranks).flatMap(lambda (page, (neighbors, rank)): computeContribs(neighbors, rank))
    ranks=contribs.reduceByKey(lambda v1,v2 : v1+v2).map(lambda (page, contrib) :(page, contrib * 0.8 + 0.2))

for rank in ranks.collect(): print rank
sc.stop()
```

Problem 2

(a) Please refer to the SVN repository

(b) Spark keeps track of each RDD's lineage to provide fault tolerance and by default every RDD operation executes the entire lineage. However if the lineage gets too long, the stack may overflow. Checkpointing is a process of truncating RDD lineage graph and saving it to a reliable distributed (HDFS) or local file system (copied from https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-checkpointing.html). Checkpointing hence improves the performance by saving the recomputation time for lineage recovery.

Checkpointed RDDs can be stored either in HDFS or local file system, which is up to the developer to decide. If running on a cluster, RDDs must be checkpointed to a HDFS path.

For this homework, *PageRank.pyspark* is executed on local mode and the checkpointing path is *file:/home/cloudera/cse427s/hwM/californiacheckpoint*. The command to set up the checkpointing directory can be found in problem part (c).

(c) Execute the SPARK application in local mode

- Commands:

  Running the pyspark shell in local mode:

  *pyspark --master local[1]*

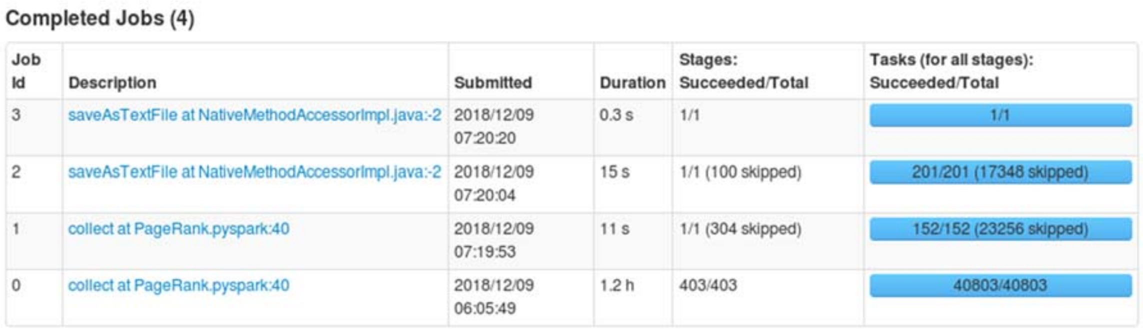
  Setup the *SparkContext* variable for local checkpointing directory:

  *sc.setCheckpointDir("file:/home/cloudera/cse427s/hwM/californiacheckpoint")*

  Execute the .pyspark application:

  *exec file("PageRank.pyspark")*

- One core is used in the execution which is specified by the --*master* option *local[1]*
- The execution detail can be found in the Spark application history server and below is the screen shot of the application history. Job 0 to 4 represent the PageRank execution, sorting the result, saving unsorted result and saving sorted result respectively. In terms of PageRank execution, there are 403 stages and 40803 tasks.



**Completed Jobs (4)**

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 3 | saveAsTextFile at NativeMethodAccessorImpl.java:-2 | 2018/12/09 07:20:20 | 0.3 s | 1/1 | 1/1 |
| 2 | saveAsTextFile at NativeMethodAccessorImpl.java:-2 | 2018/12/09 07:20:04 | 15 s | 1/1 (100 skipped) | 201/201 (17348 skipped) |
| 1 | collect at PageRank.pyspark:40 | 2018/12/09 07:19:53 | 11 s | 1/1 (304 skipped) | 152/152 (23256 skipped) |
| 0 | collect at PageRank.pyspark:40 | 2018/12/09 06:05:49 | 1.2 h | 403/403 | 40803/40803 |

- Two stages are executed in one PageRank iteration.

(d) The top 10 web ranks are:

1. (u'http://search.ucdavis.edu/', 2.7179701312864997)
2. (u'http://www.ucdavis.edu/', 2.6726023913869668)
3. (u'http://www.uci.edu/' 1.9140098401027887)
4. (u'http://www.california.edu/', 1.9010043797195546)
5. (u'http://www.berkeley.edu', 1.8805114451676075)
6. (u'http://www.berkeley.edu/', 1.8297328124237255)
7. (u'http://www.lib.uci.edu/', 1.7965400042327497)
8. (u'http://spectacle.berkeley.edu/', 1.5138475332389938)
9. (u'http://www.csun.edu/', 1.4594594594594557)
10. (u'http://vision.berkeley.edu/VSP/index.shtml', 1.4367704032532433)

The bottom 10 web ranks are (12 webpage shares same lowest rank):

1. (u'http: http://home.earthlink.net/~rberg/santiagopeak.html', 0.1569093442646898)
2. (u'http://home.earthlink.net/~rberg/blueridge.html', 0.1569093442646898)
3. (u'5http://home.earthlink.net/~rberg/screen.html', 0.1569093442646898)
4. (u'http://home.earthlink.net/~rberg/sangorgonial.html', 0.1569093442646898)
5. (u'http://www.ultranet.com/~sstv', 0.1569093442646898)
6. (u'http://www.mindspring.com/~rwf/ballomin.htm', 0.1569093442646898)
7. (u'http://www.inside-info.co.uk/scart.htm', 0.1569093442646898)
8. (u'http://www.geocities.com/Hollywood/5842/', 0.1569093442646898)
9. (u'http://www.earthlink.net/~rberg/0196/news.html', 0.1569093442646898)
10. (u'http://www.contrib.andrew.cmu.edu/org/ar99/w3vc.html', 0.1569093442646898)
11. (http://www.baycom.de/./PR/PR_main.html', 0.1569093442646898)
12. (u'http://www.rmsd.com/hamradio/corptrs.html#atv', 0.1569093442646898)

(e) The results make sense. PageRank ranks the webpage based on the quality and quantity of links to that webpage. A simple way to verify the results is to compare the number of links to the given website. Overall, webpage with higher ranks by our algorithm have more links to them on Google search engine. For example, *'http://search.ucdavis.edu/'* returns about 27,400,000 results while *'http://www.ucdavis.edu/'* returns about only 18,800,000 results. Additionally, *'http://vision.berkeley.edu/VSP/index.shtml'* no longer exists and have a significantly related results, which is about 37,600.

Problem 3: Best Location for Distribution Center

(a) The program takes in dataset orders, customers and latitude, longtitude data

i. Include only recordes from during the ad campaign (2013-05)
ii. Exclude customers who live within two-day delivery area of our current warehouse. We also exclude customers in Alaska and Hawaii because these are transported by air.
iii. Join orders with customer data
iv. delete duplicate customers
v. Join these ZIP codes with latitude and longitude data
vi. Store ZIP, latitude, and longitude in HDFS for further analysis

(b)

```
02118   42.338724      -71.07276
63139   38.610901      -90.29174
78239   29.422583       -98.56584
```

(c) Please refer to the SVN repository

(d)

```
(02118, 1059.3379591467183)
(78237, 955.8300496917677)
(63139, 423.01945102918006)
```

63139 has the lowest average distance therefore we should chose 63139.