Zhiyuan Peng

zpeng@scu.edu

Page Rank Algorithm in Spark
a brief introduction

2021-06-19

1 Introduction

PageRank is an algorithm proposed by Google to rank the searching results. It can also be used to measure the importance of the website pages. With billions of website pages in the internet, the efficiency of Pagerank becomes crucial. Spark is a very propular BigData framework which can distribute data processing tasks across multiple nodes to speed up processing tasks on very large data sets. In this project, we use spark to implement the pagerank algorithm to speed it up on Berkeley-Stanford web graph dataset.

2 Berkeley-Stanford Web Graph

Berkeley-Stanford web graph was crawled in Dec 2002 which has 685230 website pages and 7600595 links. The first four lines of this dataset is the dataset description and each of the rest lines can be split into two columns. The first column defines the start node of the link and the second is the end node. The data cleaning is easy: remove the first four lines.

3 Page Rank Algorithm

Pagerank algorithm assumes that: 1.important web pages get are more likely to be pointed to by other pages. 2. web pages are important if they are linked by important websites. We use w to represent a website, and define s_w as the rank score of the website, N as the number of the websites, d_w as the number of websites w' pointed by website w. Pagerank is an iterative algorithm which starts by initializing $s_w = \frac{1}{N}$. Then, we do the following updating until it is converged.

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w':w' \to w} \frac{s_{w'}}{d_{w'}}, \text{ where } \gamma = 0.15 \tag{1}$$

4 Parallel Compution

4.1 Dataset to RDD

To utilize the parallel computation, our first step is to load the dataset to RDD:

```
filepath = "/home/zhiyuan/Documents/BIgData/FinalProj/web-BerkStan_clean.txt"
from_to = sc.textFile(filepath)
```

4.2 Extract the Links

Each line consists of two nodes seperated by tap and the first node points to the second node. We extract the links by use function $lambda$. After $groupByKey()$, for each started node, we have a list of ended node pointed by the started node. We can think $links$ has this data structure: $[(w' : [w_1, w_2...]), ...]$

```
links = from_to.map(lambda link: (link.split("\t")[0], link.split("\t")[1])).
    groupByKey()
```

4.3 Initialize s_w

$sw\_init = 1.0/685230$ initialize the s_w for each webpage w'. We use variable $ranks$ to store all the s_w. We can think the $ranks$ has this data structure: $[(w', sw\_init)]$.

```
ranks = links.map(lambda web_neighbors: (web_neighbors[0], sw_init))
```

4.4 Calculate $\sum_{w'} \frac{s_{w'}}{d_{w'}}$

We can think $links.join(ranks)$ has this kind of data structure:$[(w', ([w_1, w_2, ...], sw))]$. Function cal_add_item calculate the $\sum_{w'} \frac{s_{w'}}{d_{w'}}$ for each w in $[(w', ([w_1, w_2, ...], sw))]$.

```
def cal_add_item(urls, sw):
    dw = len(urls)
    for url in urls:
        yield url, sw / dw

toadd = links.join(ranks).flatMap(
        lambda from_to_sw: cal_add_item(from_to_sw[1][0], from_to_sw[1][1]))
```

4.5 Update s_w

$Ranks$ has this kind of data structure: $[(w, s_w)]$.

```
ranks = toadd.reduceByKey(add).mapValues(lambda rank: rank*0.85 + 0.15*sw_init)
```

4.6 Convergence

Remeber $Ranks$ has this kind of data structure: $[(w, s_w)]$. We can join the old $Ranks$ ans new $Ranks$, then only take out the values which have this kind of data structure $[(oldvalue, newvalue)]$, finally we calculate the $\sum_w |(s_{wold} - s_{wnew})|$

```
e = old_ranks.join(ranks).values().map(lambda old_new:abs(old_new[0] - old_new[1])
    ).reduce(lambda x, y: x + y)
```

We can set the threshold, so that once the $e \leq threshold$, we can stop the iteration. In this program, I just do 20 iterations and print the e for each iteration.

5 Results

We can save the $Ranks$ by:

```
ranks.saveAsTextFile("/home/zhiyuan/Documents/BIgData/FinalProj/iteration10")
```

At each iteration, we plot the $\sum_w |(s_{wold} - s_{wnew})|$ to check whether pagerank algorithm is conveged or not.

```
  5%|▌         | 1/20 [00:46<14:39, 46.28s/it]error of iteration 0 is 0.9548739674814287
 10%|▉         | 2/20 [01:15<10:52, 36.24s/it]error of iteration 1 is 0.535223800358871
 15%|█▍        | 3/20 [01:43<09:13, 32.59s/it]error of iteration 2 is 0.2716691440456429
 20%|█▊        | 4/20 [02:12<08:17, 31.12s/it]error of iteration 3 is 0.14354045718042863
 25%|██▎       | 5/20 [02:40<07:32, 30.13s/it]error of iteration 4 is 0.08327263956703547
 30%|██▋       | 6/20 [03:09<06:55, 29.66s/it]error of iteration 5 is 0.054803646669310625
 35%|███▏      | 7/20 [03:37<06:19, 29.16s/it]error of iteration 6 is 0.03940791646409884
 40%|███▌      | 8/20 [04:07<05:52, 29.35s/it]error of iteration 7 is 0.02931532617136412
 45%|████      | 9/20 [04:38<05:27, 29.74s/it]error of iteration 8 is 0.02263870850950033
 50%|████▌     | 10/20 [05:08<05:00, 30.02s/it]error of iteration 9 is 0.017846624861976454
 55%|████▉     | 11/20 [05:40<04:34, 30.55s/it]error of iteration 10 is 0.014329756480761676
 60%|█████▍    | 12/20 [06:13<04:09, 31.19s/it]error of iteration 11 is 0.011645446222237838
 65%|█████▊    | 13/20 [06:45<03:41, 31.59s/it]error of iteration 12 is 0.009513368740565027
 70%|██████▎   | 14/20 [07:18<03:12, 32.02s/it]error of iteration 13 is 0.007817124275276819
 75%|██████▊   | 15/20 [07:53<02:44, 32.91s/it]error of iteration 14 is 0.00644460645327416
 80%|███████▏  | 16/20 [08:29<02:14, 33.67s/it]error of iteration 15 is 0.005334036795186271
 85%|███████▋  | 17/20 [09:03<01:41, 33.93s/it]error of iteration 16 is 0.004422719210952394
 90%|████████  | 18/20 [09:38<01:08, 34.28s/it]error of iteration 17 is 0.0036797252787356062
 95%|████████▌ | 19/20 [10:15<00:34, 34.94s/it]error of iteration 18 is 0.00306656068371414
100%|█████████▉| 20/20 [10:51<00:00, 32.57s/it]
error of iteration 19 is 0.002562142014988604
```

Figure 1: Print the $\sum_w |(s_{wold} - s_{wnew})|$ at each iteration

6 Conclusion

From 2, we can see, the error decreases with the iteration number increasing. From iteration 14, the decreasing almost stops, so, we can think it is converged at iteration 14.

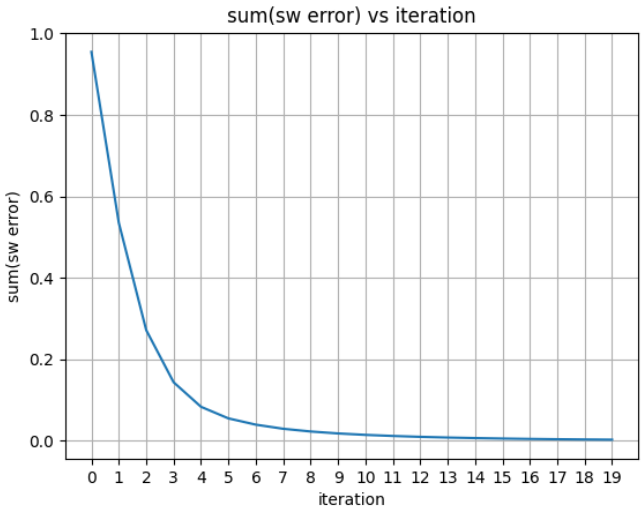

Figure 2: Plot $\sum_w |(s_{wold} - s_{wnew})|$ vs iteration number