# Part 2 Design highlights

In this part, we will discuss features we developed to enhance the search engine. In back end, we used Mapreduce to calculate the PageRank (on Hadoop).

## Calculating PageRank using Mapreduce(Hadoop)

### Motivation

Google introduced Mapreduce to calculate Mapreduce on billions of webpages more than 100 times everyday. It’s impossible to fit all the data on single machine, because of limited compute resource. Google simply distribute work to thousands of cheap computers. The Google file system is a distributed file system which is faults tolerant. In this lab, we used a popular Mapreduce Hadoop to implement a simple version of PageRank, and we actually ran the application on Hadoop

### Resources(reference)

This article introduced how does Google calculate PageRank using Mapreduce

**[1]** http://www.cs.utah.edu/~jeffp/teaching/cs5955/L24-MR+PR.pdf

This article shows data structure and Mapreduce pseudo code.

**[2]** <https://code.google.com/p/joycrawler/downloads/detail?name=Readme-0.20.0.pdf&can=2&q>=

This article depicts how to write Hadoop application in python. The mapper and reducer take advantage of Hadoop streaming feature. All the input/output are stdin/stdout, and handled by Hadoop streaming automatically. The article also gave a simple way to test mapper/reducer function on command line

**[3]**<http://www.glennklockwood.com/di/hadoop-streaming.php#wordcount:shuffle>

This article introduced how to install Hadoop on Ubuntu.

**[4]** <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/>

### Data structure

We will describe our own implementation of data structure based on article [1] and [2]. We will walk through a simple 4x4 example to illustrate our design.

For example we have the following graph:

**Matrix 1**

M = [ 0 1/3 0 1/2]

[ 1/2 0 0 0 ]

[ 0 1/3 0 0 ]

[ 1/2 1/3 1 1/2 ]

We can save it as a sparse matrix:

**Matrix 2**

1 2 4 # Page id= 1 has two links that point to page 2 and page 4

2 1 3 4

3 4

4 4 1

The first column represents each page id and followed by out link id (see above comment).

This data structure works well with our crawler, because when it crawls on a page, crawler simply output page id and followed by out links id (or saved to database directly)

### Mapper

Each page has a PageRank score (denote as pr), and each page has k numbers of out links. So for each out links on a specific page, it will has a pr \* 1/k probability to be clicked. For example, initially pr=1/4(where 4 is total page number) for matrix 2, so P(page2)=1/4\*1/2, and P(page1)=1/4\*1/2.

### Reducer

Mapper out put multiple P with the same id. The reducer aggregates all the P associate to the same page id, and then calculate

pr = beta\*(p1+p2…)+(1-beta)\*1/n

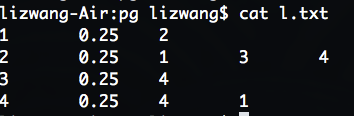
Where beta is the damping factor=0.85, and n is total number of page ids

### Iteration

Each Mapreduce task yields a set of PageRank score. We can simply iterate the Mapreduce procedure until PageRank score converge (article [2] mentioned that 15 iteration is good enough for most cases; for Google maybe 30+ iterations).

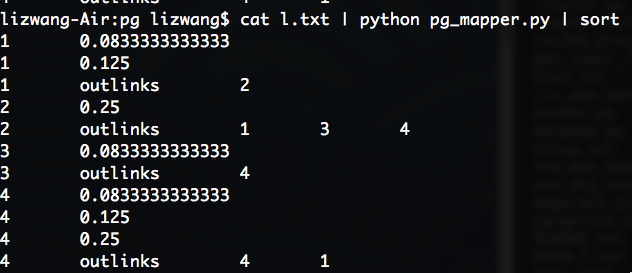
### Implementation

Firstly, we need to improve the data structure for coding convenience purposes. We improve matrix 2 to the following:



Where the second column is the PageRank score for the current page. As we mention before, initially we set PageRank = 1/n where n is the total number of page ids.

Then the mapper will output following data:

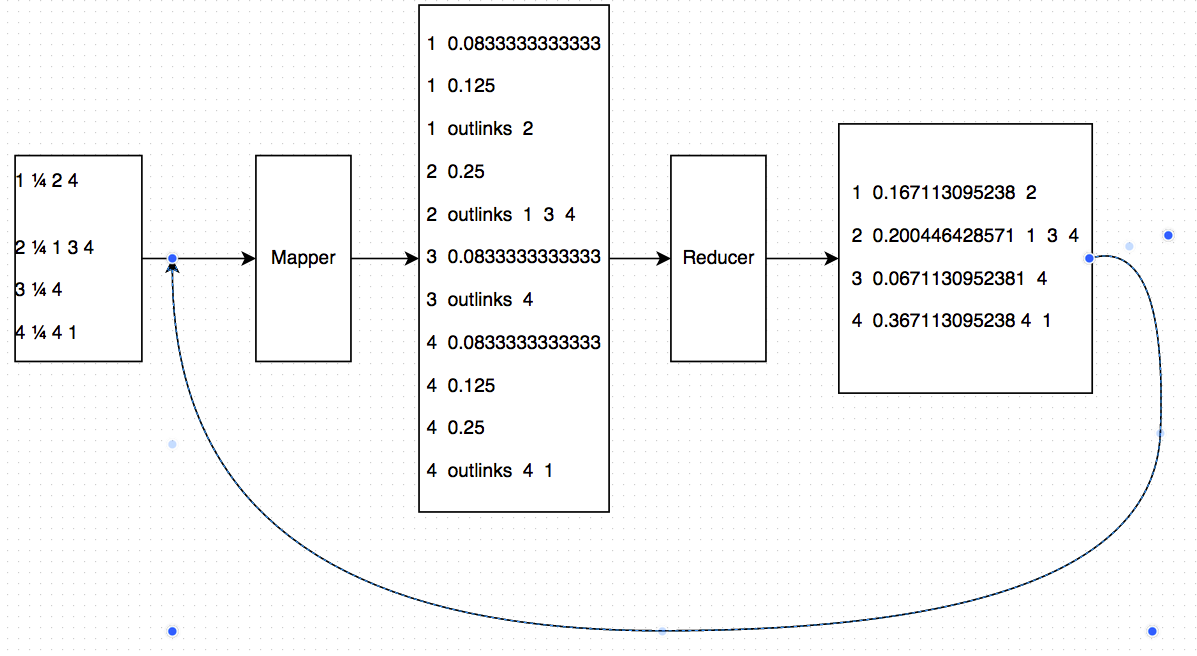


Where first column is page id, and second column is the probability (see mapper step above)

The 3rd row represent page id and out links relationship, so the reducer can output data, which can be fed to mapper in the next iteration.

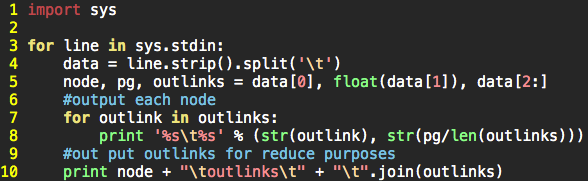
The reducer consumes mappers output, and then out a set of PageRank score which can be fed to next Mapreduce iteration

The Mapreduce task works as following:

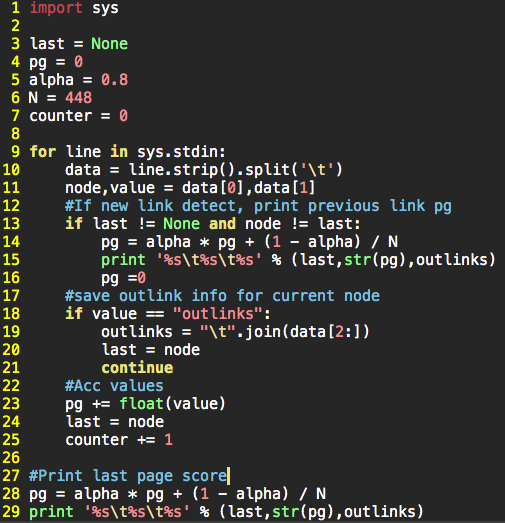


### Mapper/Reducer in python

pg\_mapper.py



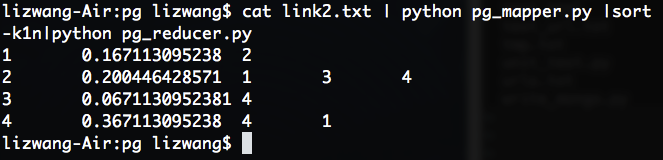
pg\_reducer.py



### Testing on command line:

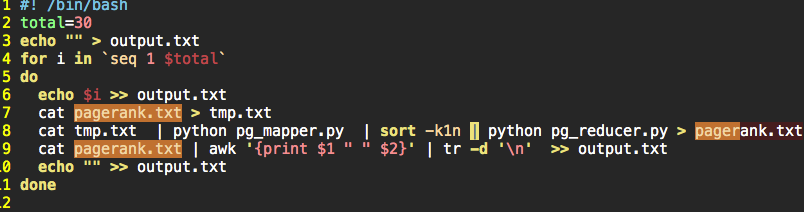
Article [3] showed an elegant way to test mapper and reducer in classic word count example on command line. We test our code by using this method. We will use the 4x4 matrix we described above as the input.

First Mapreduce iteration:



Note: We sorted the mapper’s output by the first column(sort -k1n). In Hadoop, mapper will sort output by key(first column) automatically

Next, we iterate Mapreduce task 30 times. We accomplish this in a simple bash script:



We can notice that after 20 iterations, PageRank converges (first 5 decimals)

