Hadoop PageRank (Project 2)

Technical Report

**Authors**:

Nilima Sahoo (UID# 0003644966) – Initial Draft

Hemanth Sharma (UID# ) – Review and Editing

# Introduction

The web search engine is a typical distributed system on the Internet. It is designed to search for information on the World Wide Web. The search results are generally presented in a list of results and are often called hits. PageRank is a well-known web graph ranking algorithm that helps Internet users sort hits by their importance.

# System Overview

PageRank calculates a numerical value for each element of a hyperlinked set of webpages, which reflects the probability that a random surfer will access that page. The process of PageRank can be understood as a Markov Chain which requires iterative calculations to converge. An iteration of PageRank calculates the new access probability for each webpage based on values calculated in the previous iteration. The process will repeat until the number of current iterations is bigger than predefined maximum iterations, or the Euclidian distance between rank values in two subsequent iterations is less than a predefined threshold that controls the accuracy of the output results.

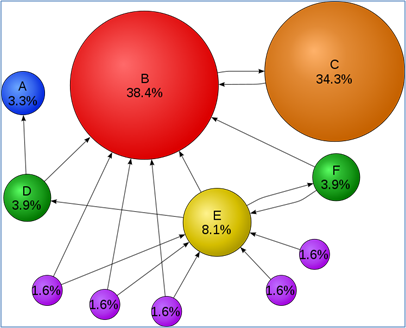
  
*Figure 1.* Mathematical PageRank for a simple network in Wikipedia

Figure 1 shows a web graph consisting of 11 vertices {A, B, C, D, E, F, G1, G2, G3, G4, G5}. Each vertex refers to a unique webpage, and the directed edge means there is one link from the source webpage to the target webpage. The percentage on each vertex represents the rank value of each webpage.

**Formula**

Eqn.1 is the formula to calculate the rank value for each webpage. We will learn this formula by applying it to the case in Fig.1. There are 11 webpages in Fig.1, which include: {A, B, C, D, E, F, G1, G2, G3, G4, G5}. Assuming the probability distribution for a web surfer accessing all these 11 pages in current iteration is {PR(A), PR(B), PR(C), … PR(G5)}, then the probability for the surfer to access Page B in the next iteration is:   
PR(B) = PR(D)/2 + PR(E)/3 + PR(F)/2 + PR(C) + PR(G1)/2 + PR(G2)/2 + PR(G3)/2   
In a general case, the PageRank value for any page u can be expressed as:  
Eqn.1:

The vertices seen in the right of the formula contain all the webpages that point to target webpage ‘u’. The L(v) refers to the out degree of each webpage in the vertices set. The initial rank values of each webpage, like PR’(u), can be any double value. After several iteration calculations, the rank values converge to the stationary distribution regardless of what their initial values are.

**Damping factor**

The PageRank theory holds that even an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d. Various studies have tested different damping factors, but it is generally assumed that the damping factor will be around 0.85. The formula considering damping factor is shown in Eqn.2. N refers to the total number of unique urls.

Eqn.2:

## System Architecture

Data flow diagram of this system is as below:

##### 

This data flow can be divided into below three parts

* Create Graph (No Code changes): Add one column, 'initial PageRank value', to the input PageRank adjacency matrix (AM). Then pass it to the PageRank program to calculate the PageRank values.
* PageRank: Took the transformed AM matrix and calculated PageRank values for all pages. Changes done in PageRankMap.java and PageRankReduce.java files.
* Clean-up Results (No Code changes): This function cleans up the targetUrls column and gives output for <sourceUrl, PageRank value> as the final result.

**Map Structure is as follows**:

* **I/P**: <key, value> = <sourceURL, pagerank#outbound urls>
* **O/P**: <key, value> = <targetURL, rankValuePerTargetURL> and <sourceURL, targeturls>

**Process:**This Map/Reduce block runs for every iteration. Based on the input url, map generates two types of <K, V> pairs.

First the <targetURL, rankValuePerTargetURL> pair will be generated. For this, it will compute the distribution of SourceURL, by dividing its PageRank value to number of outbound links. So each outbound links of input SourceURL will become key and value of this key is the computed PageRank value of SourceURL.

Apart from this pair, it will generate one more key/value pair where SourceURL is key and its outbound urls are as value. These target URLs are # separated

In case of Dangling Nodes, mapper will scatter its PageRank value to all the Nodes. So, it will generate pair in which each of the URL, including SourceURL become Key, and Value of this key is the value of this dangling node PageRank value.

In Java, PageRank Map is implemented in the following way

***<Input Key, Input value, Output key, Output Value > = <LongWritable, Text, LongWritable, Text >***

**Computational Process**:

* ***Computation****: no mathematical computation happens here, only <key, value> pair is generated*
* ***Storage****: Each <key, value> pair is stored locally where Map runs.*
* ***Replication factor****: Since working on VM, replication factor doesn’t have advantages of redundancy. Replication factor = 0*

**Reducer Structure is as follows**:

* **I/P**: <key, value> = <targetURL, rankValuePerTargetURL> and <sourceURL, targeturls>
* **O/P**: <key, value> = <sourceURL, sumofPagerank#outbound urls>

In the next function, recuder will do the addition of all values of a particular key. It skip the values which are in format of outbound links in the calculation of PageRank sum.  
  
Later, reducer joins the computed PageRank value and outbound/target URLs. It will make SourceURL as key and this combined text as value. Once this is done, it is end of one iteration. Now output of this PageRank reducer becomes the input of PageRank Map for next iteration and same process continues. This way many iterations can be run and ranking of pages could be done.

PageRank can be computed by using below formula.

At the end of last iteration, output of this reducer will be used for Cleanup Result block input. In Java, PageRank Reducer is implemented in the following way

<Input Key, Input value, Output key, Output Value > = <LongWritable, Text, LongWritable, Text >

**Computational Process:**

Computation: Mathematical addition of all values of a particular key,   
Storage: Final output is stored in HDFS  
Total number of Reducers: In this problem, only one reducer is used

# Output

##### Just run the Shell script 'compileAndExecHadoopPageRank.sh' with respective arguments. It will generate the output in HDFS output folder

##### Code is run on the given input file for 10 iterations. Output is available inside directory HadoopPageRank/output. File name is "part-r-00000".

##### Apart from that, file for only top 10 ranked list of URLs and their page ranks is also located in the same directory HadoopPageRank/output with name “output.txt”

##### 