Group 4

B649 : Cloud Computing

February 13, 2015

Project 2

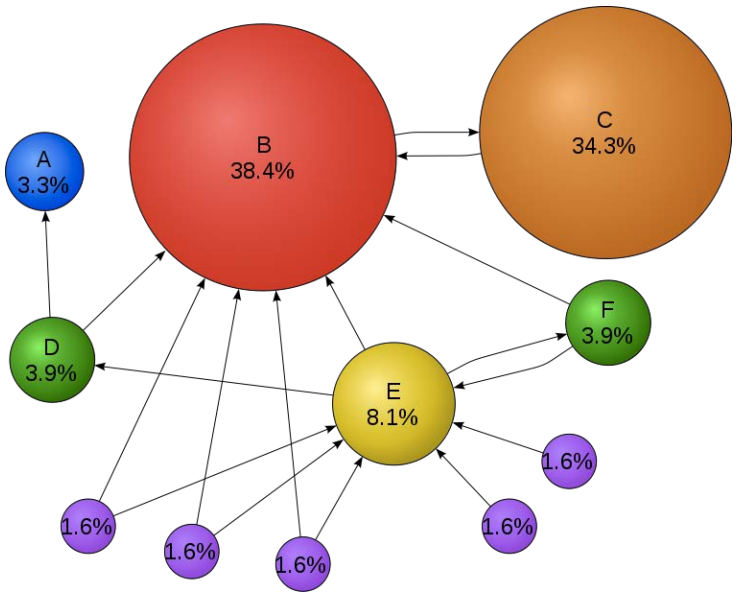
Hadoop PageRank

Page Rank

PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.[1]

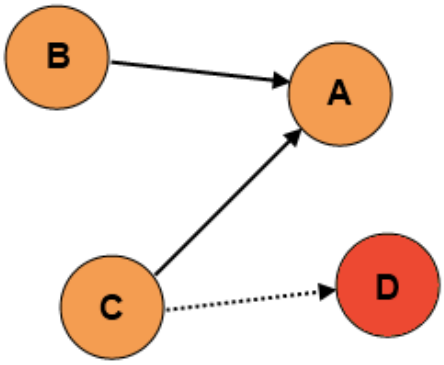
It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known.



In the above case, node B(page B) has a large page rank as many other pages have outward links pointing towards Page B. On the other hand node C has a large page rank as it is being pointed towards by a page with a large page rank (Node B).

Here is a basic example which gives an idea of how the Hadoop Page Ranking algorithm works.

Consider 4 pages: A, B, C and D (non-existing page). Page D has not been created yet, but it has an outward link from page C. The links between the pages are as follows:



In this case, page A has the highest rank, as it has 2 other pages (B and C) pointing towards it. The following formula can be used to calculate the page rank:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

d : Damping factor.

The PageRank theory holds that 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 set around 0.85.

Working of Hadoop PageRank Algorithm

Creating <Key, Value> pairs

In the Hadoop PageRank algorithm, we don't only want to create the <Key,Value> pairs, we want to create <Key,Value> pairs that include nodes, dangling nodes, page rank values and links.

All the web pages and their links can be represented as a directed graph where each vertex is a unique webpage and the directed edge indicates there exists a link from the source page to the target page. Generally this information is represented in the form of an Adjacency Matrix. In order to use map reduce to implement page rank algorithm we provide this adjacency matrix stored in HDFS, as an input to the map function. The map function requires the input to be represented in the form of a <key,value> pair.

Nodes (normal and dangling) can be represented as keys. The webpage names or the unique IDs can be considered to be keys. It is easy to differentiate dangling nodes from normal nodes in the adjacency matrix. Since dangling nodes have an out degree of 0, all the entries of the dangling nodes in the adjacency matrix would 0, whereas normal nodes would have 0 values, but they would also have a non-zero and non-negative values.

The calculated value for each page is the combination of the current page rank value and the entire outward link information for that page. This information can be differentiated with the help of any separator. Therefore, in this case, a <key,value> pair can represent a lot of important information like the source and the target URLs, page rank values, edges between the nodes and also the dangling nodes.

Dangling node are nodes which have an outward degree of 0. If we try to use the same page rank formula then while calculating the page rank value we basically divide by 0. Rather than using that approach, it is easier to just scatter the rank values of the dangling nodes to all the other URLs equally.

Map Reduce Tasks

Three Map and Reduce tasks occur sequentially. The output of one Map-Reduce task serves as the input the second and so on. The purpose of 3 map and reduce tasks are as follows:

MapReduce Level 1 (Create Graph)

Purpose: To transform the Adjacency matrix in to key value pairs which would serve as an input to the next map and reduce phase.

Map Input: The Adjacency Matrix representing the complete Web graph.

Map output: Add a column "initial Page Rank Value" and insert a "#" which separates the the page rank value and the out links.

Reducer: Take the input from the mapper and write it to the HDFS since the first level mapper runs on the local File System.

MapReduce Level 2 (Page Rank implementation)

*Purpose***:** Calculate the page rank values for each node iteratively

*Map***:** Take in the # separated input from the Level 1 MapReduce and calculate the page rank values of each node.

*Reduce***:** Take the input from the Map task and sum all the page rank values for each node iteratively.

MapReduce Level 3 (Format results)

Purpose: The sole purpose of this MapReduce job is to clean the results and write them in a presentable way.

Map: Since we are only interested in the WebPages and their page rank values this function just removes the target URL list, so basically it removes everything after the Page Rank values including the #.

Reduce: This further combines the result from the map and write the result as <pagerank,url>.

Difference in performance using inbound or outbound link

Basically, the target URL list is a list of outbound links originating from the source URL which is stored as the very first entry in the list. The contribution of every source to the target URL is computed and in the Reduce function, we take the sum of the page ranks contributed by every page pointing to it.

If we consider the inbound links, the representation of the source and target URLs change in a manner that the target URL list contains all the links which are inbound to the source. In that case we have to calculate the contribution of each of these target URLs to the source which involves quite a bit of computational overhead and which is very likely to cause performance issues.

Improving Performance

The main area where we can focus on improving performance is during the intermediate data handling stage. This can be done with the help of a combiner.

The intermediate values provided to it by the Map task is taken into consideration, it is aggregated, and the aggregated value is passed to the Combiner. The Combiner in turn reduces the load of the reducer. The Combiner sums up the values of a specific URL which is contributed from each of the inbound links, as calculated by a specific Map task. The part where all the summation occurs, is split between the Combiner and the Reducer. This aspect can significantly improve the performance.

References

- http://en.wikipedia.org/wiki/PageRank#Damping_factor
- http://blog.xebia.com/2011/09/27/wiki-pagerank-with-hadoop/