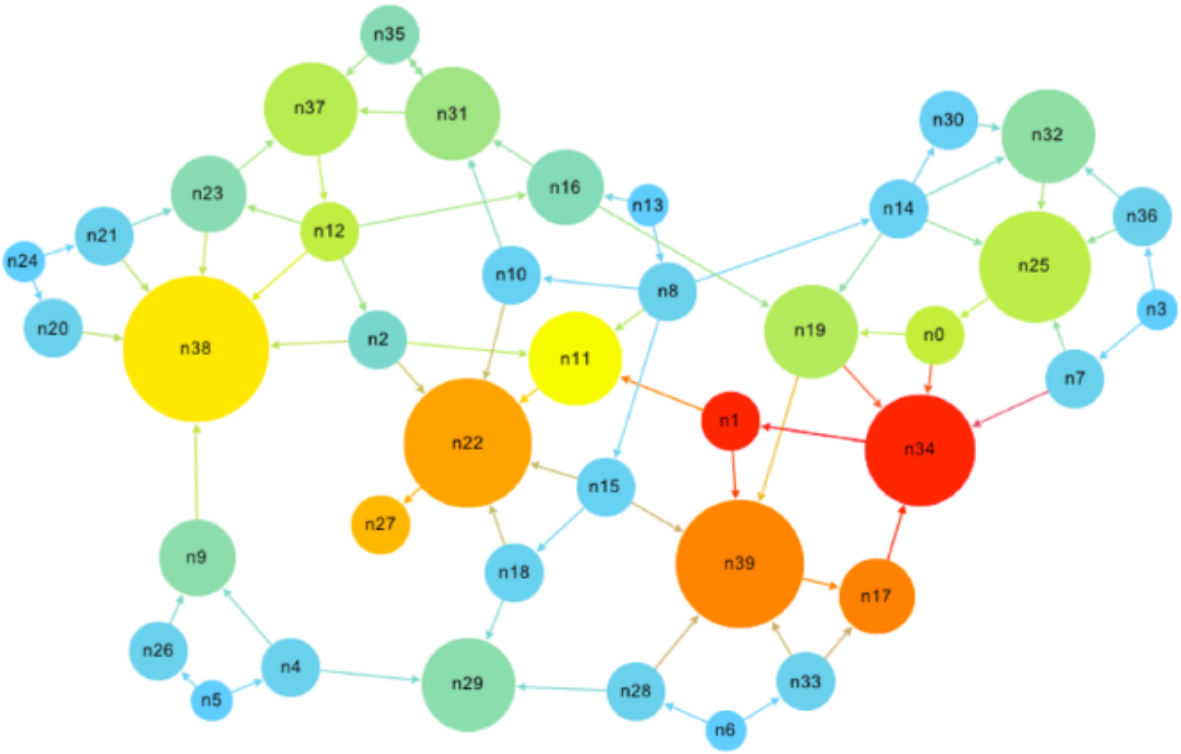
Mapreduce-Pagerank

Introduction to Parallel and Distributed Programming



04.05.2020

Prateek Garg

2017CS10360

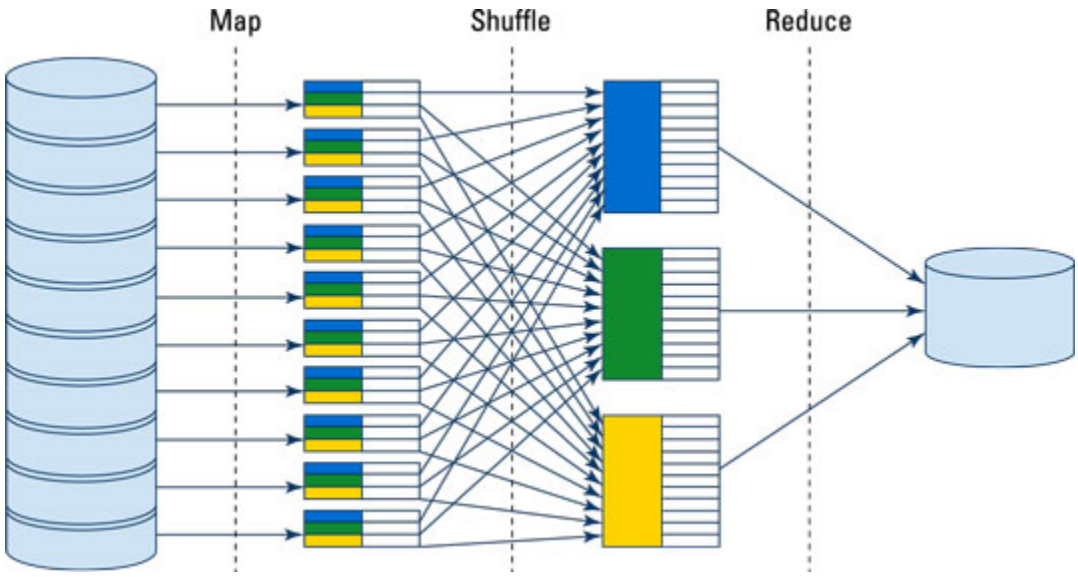
Sahil Dahake

2017CS50488

About MapReduce Paradigm

MapReduce is a programming paradigm that was designed to allow parallel distributed processing of large sets of data, converting them to sets of tuples, and then combining and reducing those tuples into smaller sets of tuples. In layman's terms, MapReduce was designed to take big data and use parallel distributed computing to turn big data into little- or regular-sized data. It works in following steps:

1. Map the data
    a. The incoming data must first be delegated into key-value pairs and divided into fragments, which are then assigned to map tasks.
    b. Each computing cluster — a group of nodes that are connected to each other and perform a shared computing task — is assigned a number of map tasks, which are subsequently distributed among its nodes.
2. Intermediate steps
    a. Upon processing of the key-value pairs, intermediate key-value pairs are generated. The intermediate key-value pairs are sorted by their key values, and this list is divided into a new set of fragments.
    b. Whatever count you have for these new fragments, it will be the same as the count of the reduce tasks
3. Reduce the data
    a. Every reduce task has a fragment assigned to it. The reduce task simply processes the fragment and produces an output, which is also a key-value pair.
    b. Reduce tasks are also distributed among the different nodes of the cluster. After the task is completed, the final output is written onto a file system.

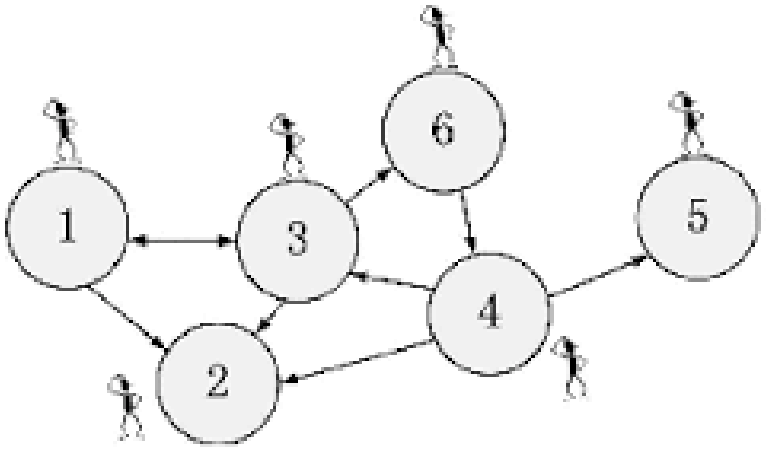


About PageRank Algorithm

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called “iterations”, through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value. The **basis for PR calculations** is the assumption that every website on the *World Wide Web* has certain importance which is indicated by the PageRank (0 being the least and 10 being the most important). The PageRank is calculated by the number and value of incoming links to a website.

1. Initially, one link from a site equaled one vote for the site that it was linked to. However, later versions of the PageRank set 0.25 as the initial value for a new website (based on an assumed probability distribution between 0 and 1).
2. The value of inbound links is determined by the amount of outbound links from the linking site and its PR. That is, because the PageRank of the linking site is divided by the total number of outbound links the page has.
3. A page that is linked to several relatively important websites (those with high

PageRank themselves) has value, while a website that has no incoming links is considered unimportant. The more the incoming links from high value websites you get, the better it is for your website.

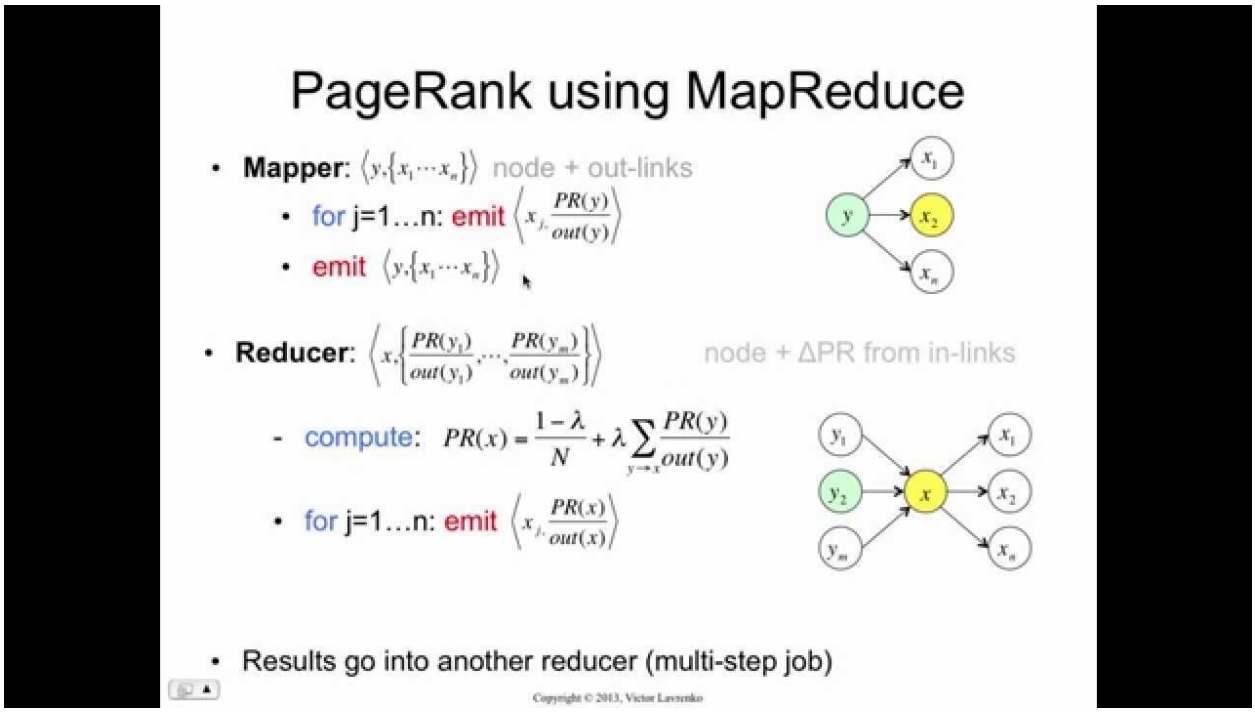


Node	Avcragc Q-PageRank
1	0.11038
2	0.22086
3	0.20294
4	0.12792
5	0.12193
6	0.21597

How can we implement PageRank using MapReduce ?

One of the most popular algorithms in processing internet data i.e. web pages is page rank algorithm which is intended to decide the importance of a webpage by assigning a weighting value based on any incoming link to that webpage. However, the large amount of internet data may lead to a computational burden in processing those page rank algorithms. Our algorithm can be decomposed into three processes, each of which is implemented in one Map and Reduce job.

1. We produce the page and its outgoing links as key and value pair, respectively, as well as total dangling node's weight and total amount of pages.
2. We next calculate the probability of each page and distribute this probability to each outgoing link evenly.
3. Each of the outgoing weights are shuffled and aggregated based on similarity of page title to update a new weighting value of each page.
4. In the calculation we consider the dangling node. In the end, all of the pages are descendingly sorted based on their weighting values. From the experimental result, we show that our implementation has output with reasonable ordering results.



Graph of PR for the following 3 implementations:-

Implementing MapReduce using C++ Library

Implementing MapReduce using our own MPI Library

Implementing MapReduce using existing MPI Library

Time taken in each case:-

Experimental Results for Latencies in each case

Reason for the order:-

Explanation for the observations

Special Remarks (if any)