Development of MapReduced Topic Sensitive PageRank

Swaraj Khadanga

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Development of MapReduced Topic Sensitive PageRank

Thesis submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Swaraj Khadanga

(Roll: 111CS0128)

under the guidance of

Prof. Manmath Narayan Sahoo

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

March' 2015

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India.

May 5, 2015

Certificate

This is to certify that the work in the thesis entitled ***Development of MapReduced Topic Sensitive PageRank*** by ***Swaraj Khadanga*** is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Manmath Narayan Sahoo
Assistant. Professor
Department of CSE, NIT Rourkela

Acknowledgment

I owe deep gratitude to the ones who have contributed greatly in completion of this thesis.

Foremost, I would also like to express my gratitude towards my project advisor, Prof. Manmath Narayan Sahoo, whose mentor-ship has been paramount, not only in carrying out the research for this thesis, but also in developing long-term goals for my career. His guidance has been unique and delightful. He provided his able guidance whenever I needed it. Yet he always inspired me to be an independent thinker, and to choose and work with independence.

I would also like to extend special thanks to my project review panel for their time and attention to detail. The constructive feedback received has been keenly instrumental in improvising my work further.

I would like to thank other researchers in my lab and my friends for their encouragement and understanding.

My parents receive my deepest love for being the strength in me.

Swaraj Khadanga

Abstract

Today Search engines are smart enough to search the content as well as it can effectively rank the fetched page(s) in an useful manner. When an user search for a content in the search engine, the search engine fetches the web pages from the database server and shows the results in an organized order according to the importance of the website/web page .The importance of a page can be calculated with a PageRank value (i.e. the number of different pages point to it). If we analyze the web a little; we can observe that it forms a sparse graph with each node representing a web page and each edge representing one hyper link. More specifically we can consider this graph to be directed. Hence the web can be represented as a matrix and PageRank can be formulated as a recursive linear equation and hence PageRank values can be calculated as an eigenvector to the equation. Spider trap and Dead end problems have been studied and those can be solved with the reformulation of web matrix with a random surfing probability also known as dumping factor. Considering these factors a map-reduce model can be developed and easily implemented in any Hadoop like environment. Map-Reduce takes the advantages of parallel processing in a cluster and sparseness of the web matrix also favors the choice of map-reduce programming model. Topic sensitive PageRank is studied and the map-reduce version for modified PageRank is designed and hence implemented.

Contents

Certificate ii

Acknowledgement iii

Abstract iv

List of Figures vi

List of Tables vii

1 Introduction 1

1.1 Motivation And Objective 2

2 Basic Concept 3

2.1 Definition Of PageRank 3

2.2 Structure Of Web 3

2.3 Mathematical Formulation 4

2.4 Use of PageRank in a Search Engine 5

2.5 Topic Sensitive Page Ranking 6

3 Literature Survey 7

3.1 Spider Trap And Dead End 7

3.1.1 Dead End 7

3.1.2 Spider Trap 9

3.2 Algorithms 10

3.2.1 Arnoldi Iteration 10

3.2.2 Power Iteration 11

3.3 Map Reduce 11

3.3.1 Programming Model 12

3.3.2 Map Reduce Model For Page Ranking Algorithm 12

4 Proposed Work **15**

4.1 Topic Sensitive PageRank / Priority Sensitive PageRank 15

4.2 Mathematical Model And Algorithm 16

4.2.1 Map-Reduce Model For TSPR 16

5 Web Crawling **18**

5.1 Parsing With Xpath 18

5.2 Data Processing 18

6 Simulation and Results **23**

6.1 Experimental Setup 23

6.2 Dataset Collection 23

6.3 Testing 24

6.4 Comparison 25

7 Conclusion and Future Work **27**

Bibliography **28**

List of Figures

2.1 A hypothetical web 3
2.2 Basic Design Blocks of A Search Engine 6

3.1 Dead End At Node C 8
3.2 Dead End at E and C 8
3.3 The reduced graph with no dead ends 9
3.4 Spider trap at node C 10
3.5 Map-Reduce Programming Model 14

5.1 Basic Design Of Web Crawler 19

6.1 Space Comparison Iterative vs Map-Reduced 25
6.2 Time Comparison Iterative vs Map-Reduced 26

List of Tables

6.1 Scraped Information 24
6.2 After Post Processing 24
6.3 Input of TSPR 24
6.4 Output of TSPR 25

Chapter 1

Introduction

Search Engines are some interesting interest of study for last decade. Most of the search engines work by crawling the web pages and then they build a inverted index by listing all the words or other strings found in which page. When a search query is fired, the terms are searched in the inverted index and all the web pages which contains the search term. A TrustRank is calculated based on several factors like the position of the search terms with in document and number of hits. After fetching the web pages and calculating the TrustRank, it is matched with the PageRank value to find the overall rank of a web page and the fetched pages are sorted and displayed according to their ranks. So a TrustRank tells us how much the web page is relevant to the search term and PageRank tells us how much the website is important in the web. For example stackoverflow.com is obviously important in web than some college's discussion forum. If we only organize with TrustRank, a highly unimportant web page, but with best suit to the search term will be populated first and those site will never lead you to other required information and hence PageRank plays an important role in case of a search engine.

The size of web is increasing on regular basis and efficient computation and processing of the such huge web is now a challenge. For last 10 years many efficient methods has been discussed and this has been a trend topic. With the introduction to map-reduce programming paradigm, this high scale data processing has been simplified with parallel processing in clusters at huge data centers. All these concepts have been studied throughout the thesis.

Apart from pageranking, topic sensitive pageranking calculation is also termed as important. As the title says, it computes biased PageRank values according to the topic of the web page. This has been a point of discussion and it has a very good impact on the results of a search engine and hence this has been studied and implemented, and compared in this thesis.

1.1 Motivation And Objective

Pageranking is an interesting topic and lots of research are going on lately. The recursive formula of PageRank can be solved with many mathematical model which can be used for solving linear equation. But the structure tells us more about eigenvector. And lots of method has been studied for solving the equation. With the introduction of map-reduce the PageRank calculations has been simplified with the use of parallel processing. Topic and Priority PageRank has been a key study and objective is to find a way to address the Topic Sensitive PageRank. Objective is to build the map-reduce algorithm for the topic/priority sensitive pageranking. All through the chapters all studied and simulations are mentioned clearly.

Chapter 2

Basic Concept

2.1 Definition Of PageRank

PageRank is a method to assign real values to web pages in the Web (the part of web that has been crawled). The logic is simple, that the higher the PageRank of a web page, the more useful and important it is. There are much many algorithm for assignment of PageRank, and Basic idea is based on a recursive formula and the variation can easily be implemented and designed from that basic idea [1].

2.2 Structure Of Web

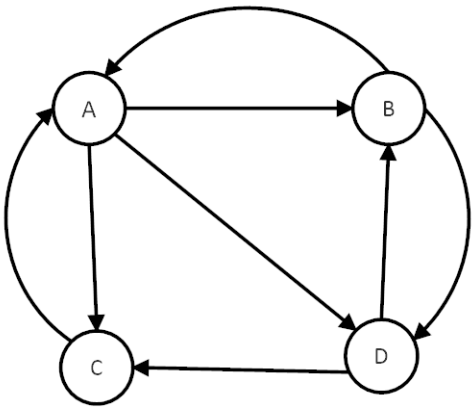

Figure 2.1: A hypothetical web

Consider the Figure 2.1, there are web pages A,B,C,D. and from A you can go to B,C,D and from B there are edges or links to A and D. This means B is a web page and there are two hyper links which takes the surfer to A and D. Similarly surfer can

only go to C from D [2].

2.3 Mathematical Formulation

Considering the Figure 2.1 again, assume our random surfer begins at web page A and it finds links to pages C, B, and D, so in the next step the surfer will jump to one of those pages with equal probability of 1/3, and has zero probability of being at A as it has no self loop. Similarly this random surfer at B can jump to A and D with half probability and to C and B with 0.

So we define this transition web matrix to explain what happens to a surfer in next step. This matrix M has N columns and rows, where N is the page count. The element m_{ij} in j^{th} column and i^{th} row has value

i. $m_{ij} = 1/z$ if page j has z edges out, and one of those pages is page i.

ii. $m_{ij} = 0$ Otherwise.

A column vector describes the probability distribution that a random surfer is at a page counted as j, and an entry at j^{th} position tell us the probability in that column vector. This idealized column vector is our required PageRank value.

We have this web with N web-pages and one random surfer is initiated at any random page with equal probability. so our start vector v_0 will contain 1/N as its entries at all the place. From the intuition of page rank we can see that the PageRank values in the next step will be the sum of incoming PageRank(s). Hence we can achieve the same with multiplying v with web matrix M. And the PageRank vector will change accordingly.

$$v = M.v \tag{2.1}$$

so we start with the vector v_0 and multiply it with M to get v_1 and then with M again to get v_2. we continue this process until v doesn't change. And that final v vector is our idealized pagerank values. Many methods can be applied to solve this equation as described in Chapter 3.

A **Markov Process** is a stochastic process which satisfies Markov Property. A process satisfies Markov property if the future of the process is predicted based on the present state of the process only. And whole Markov process is memoryless.

It is known that this distribution of the random surfer is known to approach a limiting distribution v which satisfies Equation 2.1, if following two conditions are met:

i. The graph needs to be strongly connected; which means any node can be reached from any other nodes. all pairs of nodes are reachable from one another.

ii. There should no dead ends: nodes that have no edges out. explained in Section 3.1.1.

The limit is reached when multiplying the distribution by M another time does not change the distribution. Limiting PageRank vector v is an eigenvector of M.

Eigenvector of a square is matrix is the vector when multiplied with the matrix gives us the the same vector multiplied with some scalar. That scalar is known as **eigenvalue**. In our Equation 2.1, we can clearly see that v is a eigenvector for our web matrix with eigenvalue 1.

Here the web matrix M is stochastic. so the principal eigenvector i.e. the eigenvector for largest eigenvalue is ideally our PageRank vector. so for a web matrix the stochastic property assures the highest PageRank value to be 1. Hence we apply many methods as described in Chapter 3 to find the principal eigenvector of the Equation 2.1.

2.4 Use of PageRank in a Search Engine

Each page fetched from the database of a search engine is passed through two computations: TrustRank and PageRank. TrustRank is calculated based on a lot of property of a web page, like how a search query matches with the content of a search engine. Google like search-engine uses 100s of properties to calculate the TrustRank of any web page.

After the calculation of TrustRank, it is mixed with PageRank to calculate the overall rank of a web page. Components include the presence of search terms in quality places, such as headers or the links to the page itself and number of hits to the web page etc. Based on that ranks, the web pages are sorted and displayed to the user.

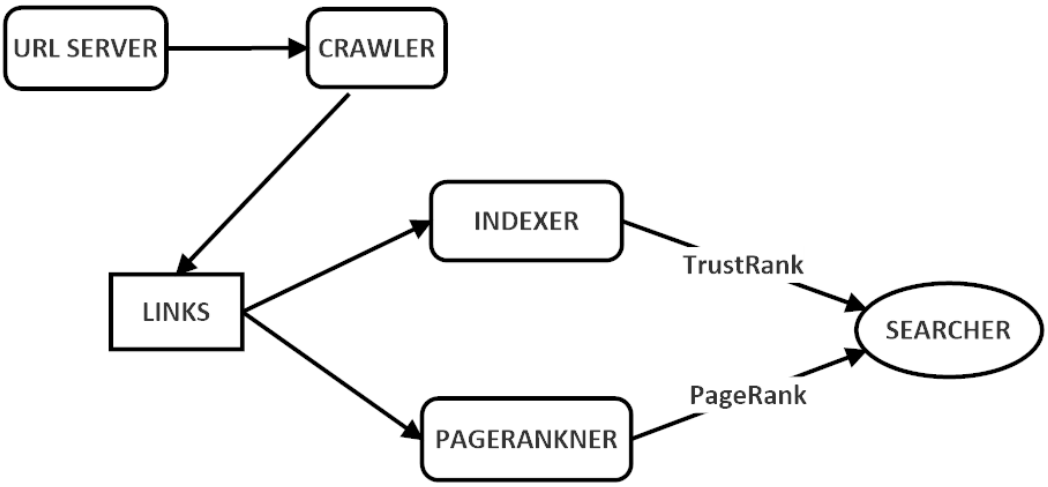

Figure 2.2: Basic Design Blocks of A Search Engine

2.5 Topic Sensitive Page Ranking

Several changes and improvements can be done to PageRank. One is that we can give more priority to certain pages and rank them more because of their topic. Searching with a search engine with pageranking works great, but sometimes same search query refers to different interests. Hence introduced topic sensitive pageranking. The best example is searching python in the web. A search query python can refer to a programming language or sometimes it can be referred to a class of snake [3]. More details is discussed in Chapter 4.

Chapter 3

Literature Survey

Equation 2.1 is a linear equation and general intuition tells us to use any method that solves a liner equation. Gaussian Elimination is one of the famous methods to solve a linear equation, but for billion nodes, Gaussian Elimination requires time cubic in the number of equations. So if we look at our web graph and its size, this method is not feasible for this situation. Iterative methods are the only way to solve these equations. our given web matrix is of size $N \times N$ where N is so huge. but we can see that the matrix is very sparse and we should take the advantage os this property. Equation 2.1 has many solutions and the stochastic property (column sum value 1) tells us that we will only get unique solution with iteration and other solutions are just scalar multiplication of this principal solution.

3.1 Spider Trap And Dead End

3.1.1 Dead End

Some web pages in the web have no out links and hence considered as dead end. If we allow dead ends, the web matrix, which is responsible for transition of random surfer is no longer stochastic, cause some columns will sum to 0 rather than summing up to 1. A matrix whose column sums are at most 1 is called sub-stochastic. If we compute $M_i.v$ for increasing powers of a sub-stochastic matrix M, then some or all of the components of the vector go to 0. After running through many iterations we won't get any information about the relative importance of pages. Figure 3.1 illustrates this clearly that we have a dead end at node C. Figure 3.2 states that we have a dead

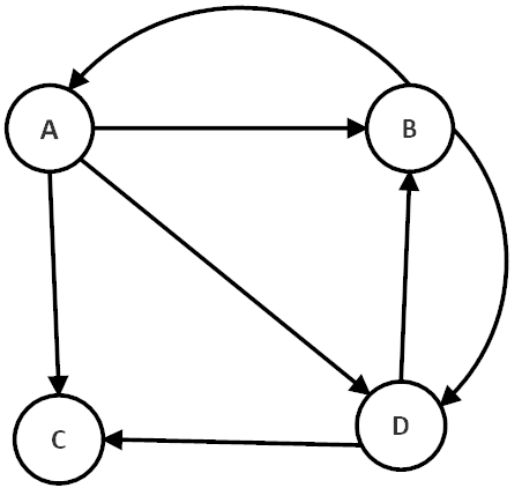

Figure 3.1: Dead End At Node C

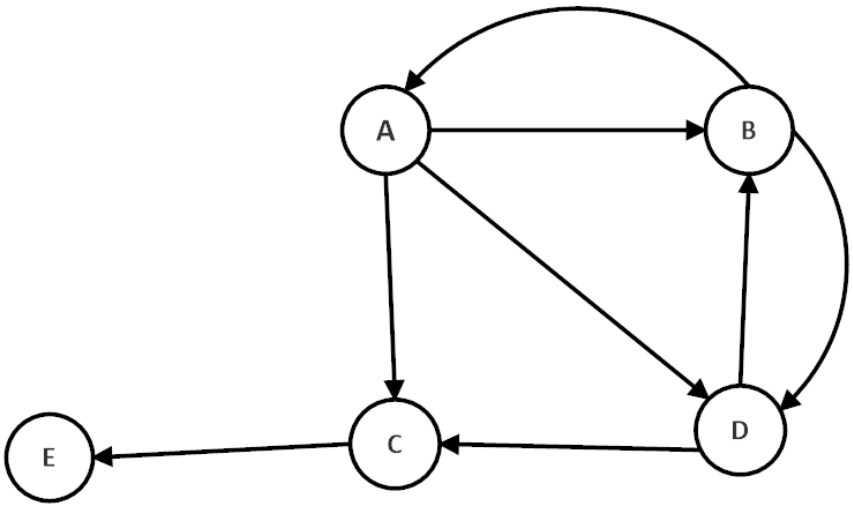

Figure 3.2: Dead End at E and C

end E and C also. If the random surfer comes to node C, then It has to go to E and stuck there.

Solution

One solution is to detect the dead ends which has no out going edges and recursively remove the dead ends until there is no dead end left in the graph. The above solution will build a smaller graph, but it makes sure to remove the dead ends from the graph. Figure 3.3 gives an illustration of the reduced graph after removing dead end problems. First remove E, it will leave C as a dead end. now again remove C then

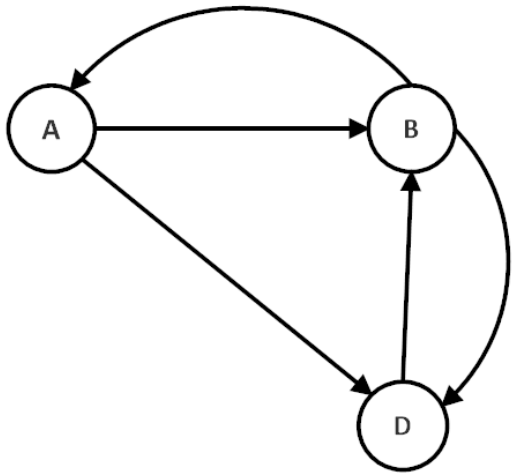

Figure 3.3: The reduced graph with no dead ends

no node will be a dead end and the remaining graph is shown in Figure 3.3. This is hard to implement for a huge graph of billion nodes, so this solution is not efficient and a new solution is needed. After addressing the problem of spider trap, a solution is discussed to solve both the problems combining.

3.1.2 Spider Trap

A spider trap is a set of nodes with no dead ends but no edge out. These web structures appears intentionally or not, but it is found frequently on the Web, and they force the PageRank calculation to put all the PageRank within that spider traps and all the PageRank value will flow to that spider trap and rank of that node will eventually 1.

Figure 3.4 illustrates this clearly.

Solution

Spider trap problem can be avoided with random tele-portation factor. what happens in spider trap is a random surfer starts surfing from any node and when it reaches to any spider trap, it gets stuck into a infinite loop and after infinite time the PageRank value of that node will eventually 1. so this solution advices us to teleport to any random node with a probability of β . on each step it will either follow a out edge or it will randomly jump to any other node in the web.

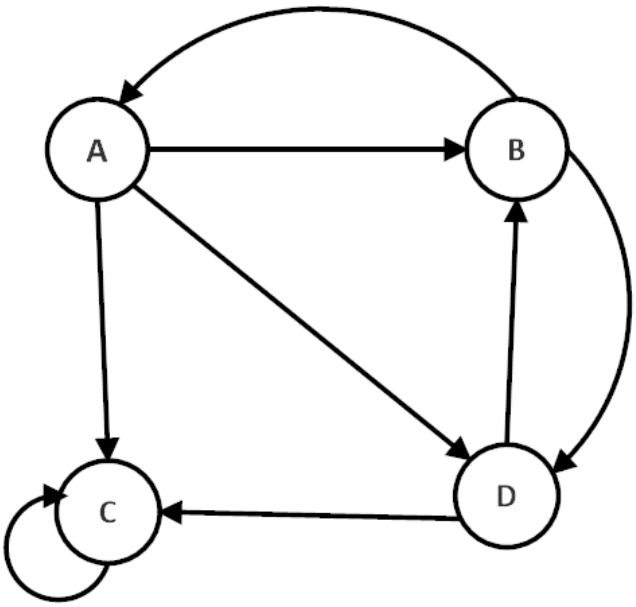

Figure 3.4: Spider trap at node C

$$v' = \beta.M.v + (1 - \beta).e/n \tag{3.1}$$

Here β is a chosen constant, and usually the value of β lies in between 0.8 and 0.9, e is a vector of all 1s with the size N, and N is the number of nodes in our graph. The term $\beta.M.v$ is for the case where, the random surfer chooses to follow an out-edge from the present page with probability β. The term (1 - β)e/n is a vector, each of whose components has value (1 - β).e/n and represents the introduction, with probability 1 - β, of a new random surfer at some random page.

3.2 Algorithms

3.2.1 Arnoldi Iteration

Arnoldi iteration is one powerful iteration used to solve the liner equation and find the eigenvector. But it doesn't take the benefit of previously known eigenvalue. However it calculates huge Hessenberg matrix and hence much inefficient for PageRank calculation. This uses the matrix for all it's computation and unlike Web-Matrix this Hessenberg matrix is not sparse, so with the increase of number of nodes, we can't use the sparseness property to design any map-reduce solution. Based on Arnoldi

process [4] many methods can be built to calculate the PageRank values. But the above mentioned limitation forces us to think beyond methods which uses Arnoldi iteration.

3.2.2 Power Iteration

Arnoldi process has better convergence compared to power iteration, but power iteration [5] suits well for map-reduce type programming model. So Power Iteration is studied, and hence its map-reduce model.And it takes the advantage of previously known eigenvalue [6]. Algorithm 1 tells us in brief about the power iteration. This

Algorithm 1: Power Iteration

Data: Web Matrix M , intial pagerank vector v_0 and tolerance eps
Result: stable pagerank vector v_0 after convergence

```
1 v=v_0;
2 v_old=null;
3 while abs(v-v_old) ≤ eps do
4     v_1=Mv;
5     v_old=v;
6     v = v_1;
```

iteration can easily be converted to a map-reduce model.

3.3 Map Reduce

Processing huge data set is a challenge, a standalone system cannot handle the computing requirement for that kind of computation. Memory and computing limitations in a single system drives us to use parallel system. But writing a simple program in old parallel system is a very tedious task. So Google came up with a very simplified programming model called map-reduce and that can be effectively used for batch processing of a huge dataset. Map-reduce is based on the concept of scaling out rather than scaling up. Adding more new system to a system is much cheaper than scaling up the performance of a system. This programming model works completely on a clustered or distributed environment [7]. User has to write a map and a reduce code and this code is executed in every single system in mapper phase

and reducer phase in the distributed file system [8]. Map-reduce model is built on the concept of distributed file system. The whole dataset is not kept in a single centralized system, rather it is distributed in all the working system. And built on the concept of moving code to the data, rather moving data to code. This effectively decreases the requirement of movement of data inside the cluster. Output of map data is written to the local disk and the reducer take data directly from the mapper. The dataset is represented as (Key,Value) [9] pair both in mapper and reducer.
Why Map-Reduce?

i. Scalability

ii. Cheap

iii. Fault Tolerance

3.3.1 Programming Model

map (in-key, in-value) -> list(out-key, intermediate-value)
Processes input key/value pair
Produces set of intermediate pairs

reduce (out-key, list(intermediate-value)) -> list(out-value)
Combines all intermediate values for a particular key
Produces a set of merged output values (usually just one)

Figure 3.5 tells us in detail about the working of algorithm.

3.3.2 Map Reduce Model For Page Ranking Algorithm

Algorithm 2: Mapper

Require: *key*[url,pagerank],*value*[outlink-list]
url: key of the input i.e. one url
outlink-list: All the urls that can be visited from the current key url
pagerank: pagerank value of the key in current iteration

```
1: for outlink in outlink-list do
2:     emit( key: outlink, value: pagerank/size(outlink-list) )
3: end for
4: emit( key: url, value: outlink-list )
```

Algorithm 3: Reducer

Require: *key*[url],*value*[list-pr-or-urls]
url: key of the input i.e. one url
$list - pr - or - urls$: PageRank value(s) of the key in current iteration or adjacency list in string format

```
1: outlink-list = []
2: pagerank = 0
3: for pr-or-urls in list-pr-or-urls do
4:     if islist(pr-or-urls) then
5:         output-list=pr-or-urls
6:     else
7:         pagerank += double(pr-or-urls)
8:     end if
9: end for
10: pagerank = 1 - β + ( β * pagerank )
11: emit( key: [url,pagerank], value: outlink-list )
```

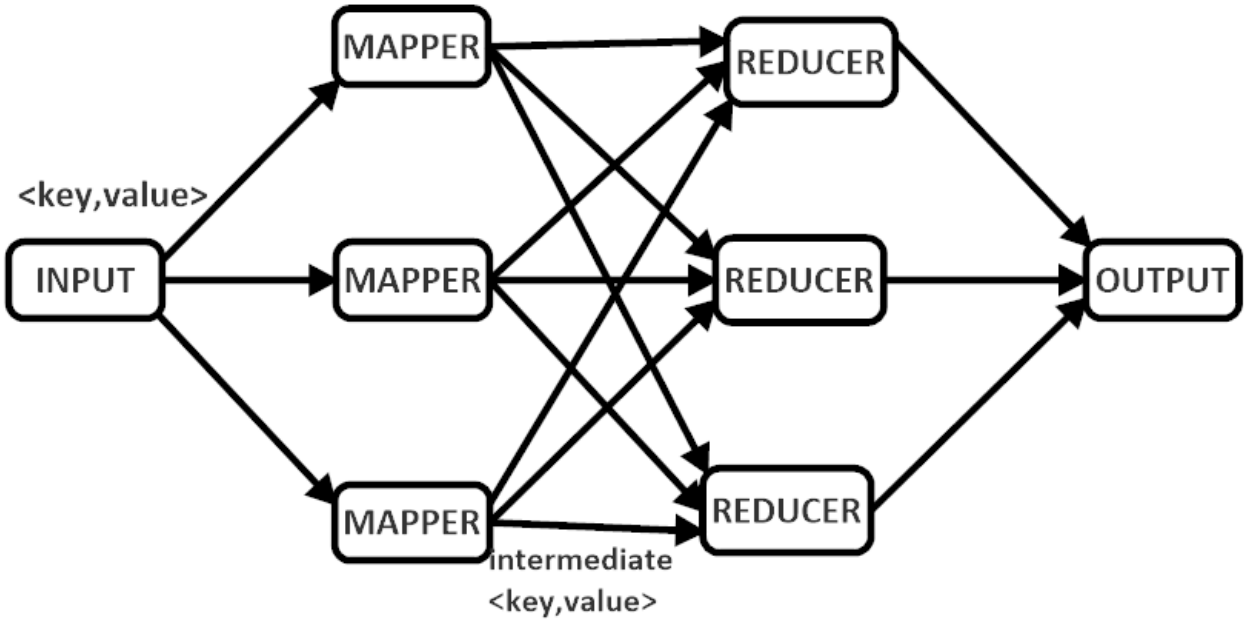

Figure 3.5: Map-Reduce Programming Model

Chapter 4

Proposed Work

4.1 Topic Sensitive PageRank / Priority Sensitive PageRank

Searching with a search engine with pageranking works great, but sometimes same search query refers to different interests. Hence introduced topic sensitive pageranking. The best example is searching python in the web. A search query python can refer to a programming language or sometimes it can be referred to a class of snake. Ideal solution is to have a PageRank vector for each user. Each user's interest can be measured by the search engine by mining the search history of the user. But there are billions of users and billions of web pages, so storing a PageRank vector for each user, which count is increasing dynamically is not feasible. So we need to do something simpler. The topic sensitive PageRank approach has the similar style but it works in a different way. Instead of storing all the PageRank vector for each user, pagerank vector for each topic is stored which decreases the amount of memory required significantly. While we surely lose little accuracy, we get the benefit by saving a lot amount of memory in this process of pageranking. Users are classified according their interest and for each search term and respective topic sensitive PageRank vector is fetched and used. From the intuition of the above concept we will teleport to only that topic sensitive node with a probability of 1- β and the equation will be modified as Equation 4.1.

4.2 Mathematical Model And Algorithm

A simple mathematical model can be built using the intuition that each time the surfer will jump to any topic sensitive node or nodes that are marked as priority.

$$v' = \beta Mv + (1 - \beta)e_S/|S| \tag{4.1}$$

given PageRank equation can simply modified as Equation 4.1. M, v, β are the previously defined factors. e_S is a vector with same length as v, defined for a specific Topic. S is the set of pages which are identified under the required topic. e_S contains 1 for those pages which are in set S and 0 otherwise.

4.2.1 Map-Reduce Model For TSPR

Algorithm 4: Mapper

Require: *key*[url],*value*[topic,pagerank,outlink-list]
url: key of the input i.e. one url
outlink-list: All the urls that can be visited from the current key url
pagerank: pagerank value of the key in current iteration *topic*: topic of the current page

```
for outlink in outlink-list do
    emit( key: outlink, value: [pagerank/size(outlink-list), topic=null] )
end for
emit( key: url, value: [outlink-list,topic] )
```

Algorithm 5: Reducer

Require: *key*[url],*value*[list-pr-or-urls,topic]
url: key of the input i.e. one url
$list - pr - or - urls$: pagerank value(s) of the key in current iteration or adjacency list in string format *topic* topic of the url

```
1: outlink-list = []
2: pagerank = 0
3: for pr-or-urls in list-pr-or-urls do
4:    if islist(pr-or-urls) then
5:       output-list=pr-or-urls
6:       topic=topic
7:    else
8:       pagerank += double(pr-or-urls)
9:    end if
10: end for
11: if topic==global-topic then
12:    pagerank = (1 - β)/|S| + ( β * pagerank )
13: else
14:    pagerank = β * pagerank
15: end if
16: emit( key: [url], value: [topic,pagerank,outlink-list] )
```

Chapter 5

Web Crawling

Web crawling is a important part for implementing pageranking algorithms. By crawling the web we can build the web graph.

i. Get a set of N seed pages.

ii. Select a Page and start crawling

iii. Crawl page P. Now parse page P using XPath or similar technologies.

iv. Extract all the links from page P.

v. Resolve relative path to absolute path.

vi. select one link and goto (iii)

5.1 Parsing With Xpath

XPath [10] are used to parse XML like file(s). HTML is similar to a XML file and we can use the same tools to parse HTML files.

5.2 Data Processing

URL-seen test

Web crawler will encounter multiple links to the same document. To avoid downloading and processing same document multiple times, a URL-seen test [11] is generally performed on each extracted link and based on the test it is decided

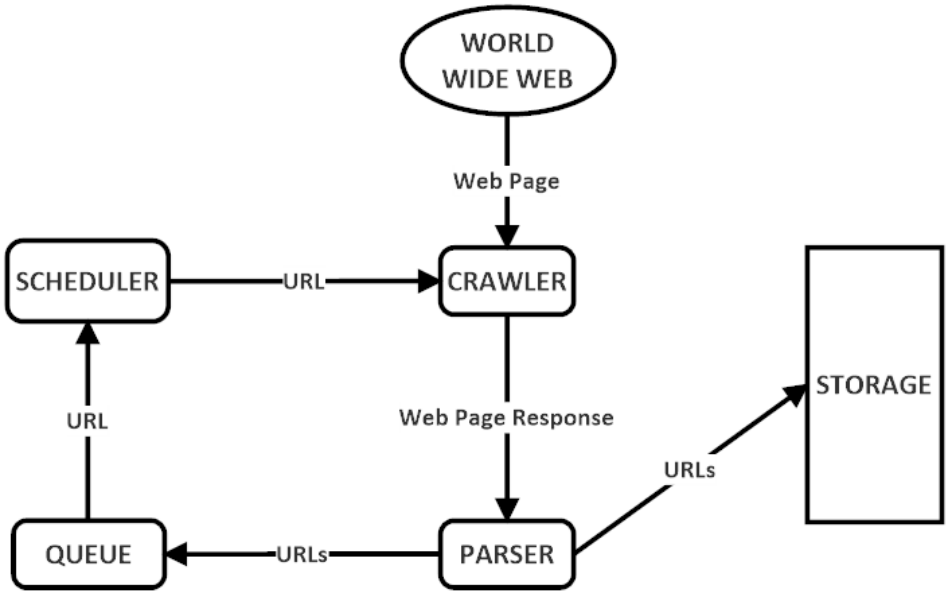

Figure 5.1: Basic Design Of Web Crawler

whether to add the link to frontier or not. Generally MD5 or SHA1 hashing is used for that.

Resolving Relative URLs

Every crawled page has a base URL.A base URL is the consistent part of your web address. Fetch the base URL of the selected page and now join the base URL with the relative URL to find the absolute path. realtive URL path is the path of the web page inside the web server or the file path of the respective web page inside the server directory. after finding the web server a web path is located within the web server and the required web page or web server is called. So during scraping we need to find the absolute path from the relative URL. And then only the page is added to the queue.

Post Processing URLs

Every URL has many parts.

i. scheme

ii. netloc

iii. path

iv. query

v. fragment

When we want to implement pageranking algorithm most of the part of the fetched URLs are useless and hence we need to process the URLs to get the meaningful part of URLs only. using any url-parser we can parse the URL and get required URL easily.

i. scheme is generally the protocol used to address the web site or network. e.g. https, ftp, http etc.

ii. netloc is the network location or the web site address of the website. For example it is the server's base address of web sites.
e.g www.google.com, www.yahoo.co.in

iii. path is the path of the web page inside the web server or the file path of the respective web page inside. after finding the web server a web path is located with in the web server and the required webpage or web server is called.

iv. query is the part of the url usually the terms used to send data to a web-page using GET request. those are usually expressed with ? and & separated string.

v. fragment is used to address to a web page. it is used to address one object with in a webpage. when it is needed to address the whole page only one # followed by space is used.

path, query and fragment are useless in this case. assume 2 pages are there in www.example.com i.e. www.example.com/a.php and www.example.com/b.php. Now we perform 2 seach queries in these web pages and passed with a GET query and move to a fragment of webpage. so assume 2 queries like

i. www.example.com\$a.php$?q=query1#m

ii. www.example.com*b.php*?q=query2#q

Now assume two queries to the web site www.example.com. Tough these two refers to two different web-pages, but these makes these add same interest to the same web-site. so keeping the search query and fragment doesn't make any sense to our PageRank algorithm. so post process it to remove the unwanted part of the URLs.
we will get same www.example.com for both crawled web page. so PageRank will be accumulated for www.example.com only. This makes more sense to our pageranking algorithm.

Input Formatting For Map Reduce

Each data scraped from the web has a format source-link,dest-link stored in a raw file. Now we have to change into a required format for our pageranking algorithm.
In order to change the format of such huge dataset we again one more map-reduce algorithms. This Mapper and reducer takes the parsed URLs and format it for the requirement input type of map reduce.

As input format is an important issue in pageranking algorithm, the above comma separated values need to be parsed and the sparse web graph need to expressed in required format. the web graph is so huge that crawled data can't be processed only with main memory, hence it leads us to think about the map-reduce design pattern.

The proposed algorithm is designed as Algorithm 6 and Algorithm 7

Algorithm 6: Mapper

Require: *key*[comma-sep-urls],*value*[blank]
comma − sep − urls: src,dest
1: src,dest=*comma − sep − urls*.split(',')
2: emit(src,dest)
3: emit(dest,blank)

Algorithm 7: Reducer

Require: *key*[url],*value*[dest-urls]
$dest - urls$: all the urls that are pointed by the current key url

```
1: pagerank=1.0
2: adj=”
3: for url in dest-urls do
4:     adj.append(url+’,’)
5: end for
6: emit(url,[pagerank,adj])
```

Chapter 6

Simulation and Results

6.1 Experimental Setup

i. Python 2.7.9

ii. JDK 7

iii. Hadoop 2.5.1

iv. Yarn

v. Ubuntu (Hadoop in pseudo distributed mode)

vi. scrapy (open source project for crawling)

vii. virtual machine

6.2 Dataset Collection

A spider is designed with SCRAPY in python and it's run against different URLs to fetch data and post processed. As mentioned above in Chapter 5, all the concepts are implemented with SCRAPY open source project. Required web sites www.nitrkl.ac.in,www.cet.edu.in,www.iiit-bh.ac.in and similar sites are crawled to collect the web graph.

Table 6.1: Scraped Information

www.nitrkl.ac.in	20965
www.iiit-bh.ac.in	16961
www.cet.edu.in	4193
...	

Table 6.2: After Post Processing

www.nitrkl.ac.in	1474
www.iiit-bh.ac.in	2046
www.cet.edu.in	227
...	

6.3 Testing

Sample Data

From the input and output tables, pr1 is the PageRank value calculated with iterative algorithm in a single machine and pr2 is the PageRank calculated in map-reduce iteration. they are found to be same. similar small test cases are taken and PageRank values are compared.

With the crawled data with around 1 lakh nodes iterative version will throw segmentation fault, but at the same time map reduce will produce result in a pseudo distributed ubuntu system. with all these testing map reduce version of the topic sensitive pageranking algorithm is proved to be correct and hence with the feature of map-reduce it can be easily scaled out.

Table 6.3: Input of TSPR

page	pr adj	topic
0	0.25 1,2,3	1
1	0.25 0,3	1
2	0.25 0	2
3	0.25 1,2	2

Table 6.4: Output of TSPR

page	pr1	pr2
0	0.332	0.334
1	0.261	0.261
2	0.186	0.184
3	0.218	0.217

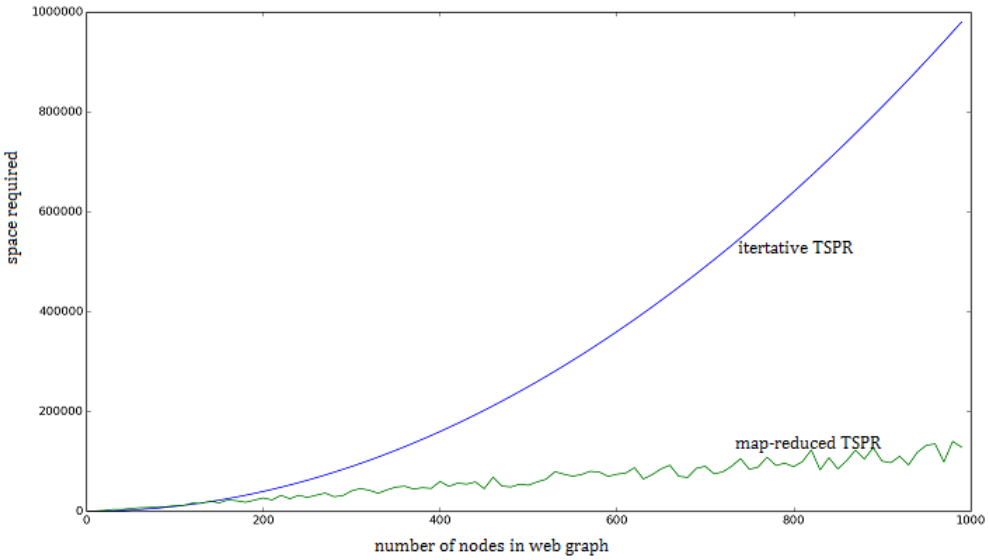

Figure 6.1: Space Comparison Iterative vs Map-Reduced

6.4 Comparison

Space Comparison

i. If we closely observe the iterative version of the power iteration we see that a $N \times N$ web matrix is required to perform the operations in the web matrix. So for a web with N nodes we need to store $N \times N$ size matrix.

ii. We store the web graph in adjacency list. So on an average a web page has 100 out-links only. that is how this representation helps us save memory. With increasing number of nodes we need to store billion of nodes, so a standalone system can't handle, but map reduce model gives us the benefit to run the same code in a cluster just with some configuration changes and data can be distributed to multiple systems.

Time Comparison

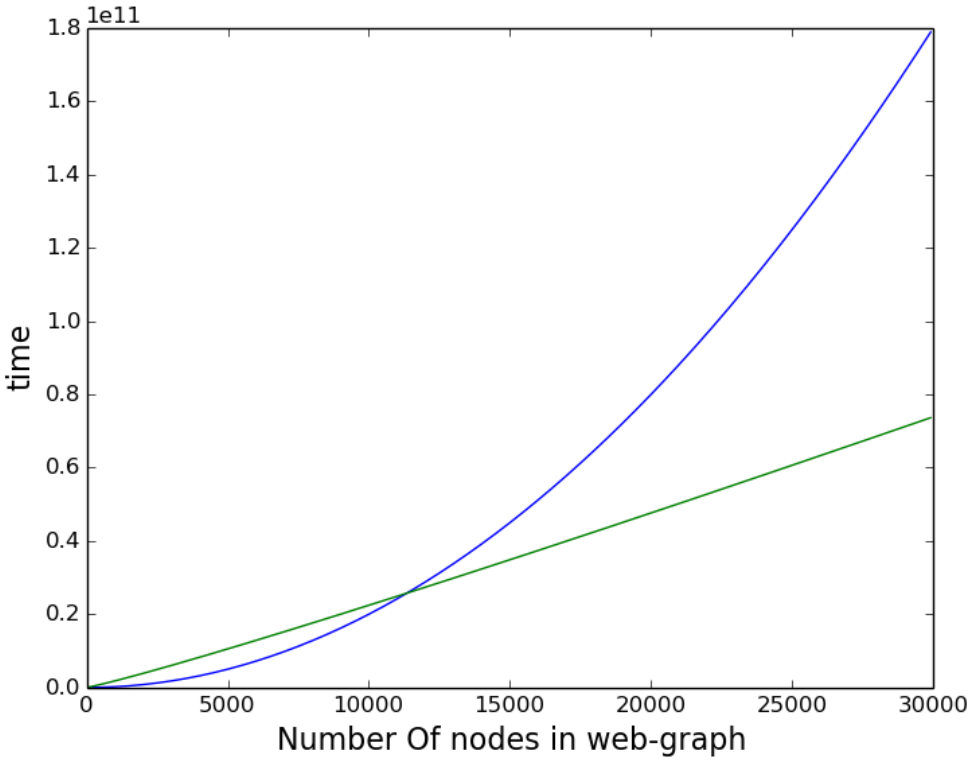

Figure 6.2: Time Comparison Iterative vs Map-Reduced

i. If we closely observe the iterative version of the power iteration we see that a $N \times N$ web matrix is required to perform the operations in the web matrix. so a vector of size N is multiplied with $N \times N$ web matrix. That takes N*N multiplications.

ii. As mentioned above we store the web graph in adjacency list. So on an average a web-page has 100 out-links only. TSPR-Mapper takes O(N), TSPR-Reducer takes O(N) and inside shuffle and sorting takes some time. Overall time graph is plotted as in Figure 6.2.

Chapter 7

Conclusion and Future Work

Based on the intuition of topic and priority sensitive pageranking a smaller change to the initial pagerank is studied. Effective calculation of pagerank is performed with mapreduce in pseudo distributed mode in ubuntu system. The MapReduce topic sensitive pageranking is proposed and hence implemented for the scraped data. the equation can be further modified for better accuracy and performance.

Bibliography

[1] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[2] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[3] Taher H Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002.

[4] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[5] Taher Haveliwala, Sepandar Kamvar, Dan Klein, Chris Manning, and Gene Golub. Computing pagerank using power extrapolation. *Stanford University Technical Report*, 2003.

[6] Nan Gong. Using map-reduce for large scale analysis of graph-based data. 2011.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] Jeffrey Dean and Sanjay Ghemawat. Distributed programming with mapreduce. *Beautiful Code. Sebastopol: OReilly Media, Inc*, 384, 2007.

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[10] James Clark, Steve DeRose, et al. Xml path language (xpath) version 1.0, 1999.

[11] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.