Implementation of Improved PageRank Algorithm on Hadoop

Simanta Sarkar
MSc. Data Analytics
National College of Ireland
Dublin, Ireland
x18201148@student.ncirl.ie

***Abstract*—The PageRank algorithm is one of the most discussed topics for processing large volume internet data. The primary purpose is to rank the Web pages through allocating weightage based on the links pointing towards the Web page to measure the importance of the same. To overcome the computational difficulty in processing the algorithm the paper proposes an improved PageRank algorithm to be implemented over a distributed environment using the Hadoop MapReduce architecture. The improved algorithm is sub divided into six process, most of which is implemented in Map and Reduce task. The final PageRank is computed based on the convergence property of Power Iteration algorithm.**

Keywords—PageRank, Hadoop, Distributed programming, Link analysis.

I. INTRODUCTION

In the recent years with the exponential growth of users and the internet data Google needs to process large volume of data every second. Here comes the importance of analyzing the big data to extract valuable information. One of the booming research topics is PageRank algorithm in processing large data. The purpose of the research is to rank the webpages to measure the importance through assigning weightage based on the incoming links to the same.

Previously, there have been several researches regarding the page rank algorithm for several purposes. For example, while a basic idea of page ranking as an algorithm for measuring the importance of a web page has been proposed, proposals has also been made for a relation-based page rank algorithm for semantic web search engines. However, as far as the awareness of the writer of this paper is concerned, there are but a few papers which have explained an implementation of simple page rank algorithm in a distributed system.

The algorithm is however having easy implementation in a single machine as little information as page link, title, id, outgoing link are required. However, processing large volume of data poses the scalability challenge such computational time.

Therefore, my paper aims to process an improved page rank algorithm over multiple machines in a distributed system. In this paper, I have used the Hadoop MapReduce framework and Python MRJob module to implement the same; the preference for Hadoop MapReduce being due to its ability to process large data over a distributed system without taking into account any issues such as job failure handling, scheduling and synchronization.

The PageRank calculation can be divided into five sub-algorithms, each of which has been executed as single map reduce job. Firstly, the raw webpage data has been parsed for extracting page title and its outgoing links like key and value. This has been followed by the link graph size calculation, initialising PageRank, the sum of dangling nodes PageRank with finally its calculation. After which, the probability of each page is calculated and distributed evenly to each of outgoing link, here the dangling node and random jumping factor has also been taken into account. The remainder of the paper is organized as follows: Section II provides a background and works in relation to the PageRank Algorithm and it is based on this model that the data in section III has been defined. Then in Section IV, the improved PageRank algorithm has been presented; followed by implementation of improved PageRank on Hadoop distributed environment, and finally the numerical results obtained from the simulations has been discussed in section V. The paper is there after concluded in Section VI.

I. LITERATURE REVIEW

A. Background

The revolutionary innovation was brought in the search engine by Google through the introduction of the 'Page Rank'. It is an algorithm to assign a real number to each web page or a part of it to rank according to their importance. However, there is no rigid algorithm for Page Ranking. With new real world challenges the algorithm is modified over time.

The challenge prevailed with applications that demanded parallel processing is greatly resolved by the emergence of the large-scale Web services. With installation of large number of independent operating compute nodes increasing number of computations can be performed. Compared to the generic special-purpose parallel machine the cost is significantly reduced for compute node installation being commodity hardware. This has added a new dimension to the programming systems with the advantage of parallelism and simultaneously eliminating the reliability problem.

[1]MapReduce, a computing style implemented in multiple system inclusive of internal implementation of Google and Hadoop which is open-source implementation coupled with HDFS file system sourcing from Apache Foundation. It comprises of two function Map and Reduce with the following discussed executional flow. The Map converts the segments sourcing from distributes file system into a progressive key value pair. The master controller collects and sort the key value pair by key followed by distributing to all Reduce task. The Reduce aggregates all the values corresponding to a key where at a particular time only one key is worked on. However, MapReduce is profitable only when the database is significantly large and updated less often. Due to the widespread influence of MapReduce it has generated extensions and modifications of somewhat similar characteristics.

In [2]Google is introduced as the large-scale search engine model that majorly utilize the hypertext architecture. It is structured to upgrade and produce improved search results

over the existing system through efficient page crawl and indexing the Web. Approximately hundred millions pages of the model with full text and hyperlinks are available at http:llgoogle.stanford.edu/. Engineering the search engine is posed with challenge of indexing over hundreds of millions of Web pages. Due to hasty advancement in technology and Web escalation building a search engine differs significantly from the former times. The research work was first ever to provide the detailed explanation of the Google search engine. To provide an improved search engine over the existing one it is challenged with the scalability and technical issues of big data and utilizing information from hypertext respectively. The paper addresses the problem providing a solution to extract information from hypertext and undisciplined hypertexts where an individual can publish unrestricted content. The link graph introduced is a graphical representation of the Web. The PageRank algorithm calculates a value which marks the importance of the Web page with respect to subjective idea of human judgement. The importance or quality of the web page is approximated through counting the incoming and outgoing links of the page. Mathematically, PageRank for a page A can be defined as,

$$\mathrm{PR(A)} = \beta\left(\frac{PR(T1)}{C(T1)} + \cdots + \frac{PR(Tn)}{C(Tn)}\right) + (1-\beta)$$

Where T_i, (i=1…n) are the number of pages citing the page A and C(A) are the counts of outgoing links of page A. The dumping factor is included as the parameter β which is usually set to 0.85.

Addition to the search quality, Google considers the scaling factor. Initially, implementing Google had experienced setbacks like CPU, memory capacity and access, disk capacity, IO of network and many more. However, evolution of Google through several operations have been able to overcome the challenges.

B. Link Graph

Analysing the Web as a graph uncovers considerable insights about Web algorithms characterizing the crawling, searching, community discovery evolution. [3]The directed graph represents static pages as the nodes and the page links as the arcs respectively. Thus, in recent research works algorithms from graph theory to address profusion of search, mining and retrieval problems. The modification of the used techniques and implementation of the methodologies of graph theory combined with both traditional and contemporary methods has proved success in examining context, content and improved browsing criteria.

C. Link Spam

The intention of link spamming is to mislead the search engine leading to a link based false high ranking for a targeted web page. [4]The introduced concept of spam mass can be defined as a measure of link spamming impact on the page ranking. The link spam farm model involving interconnected nodes associated with link spamming explains the thumb rule of the link spamming. The spam farm consists of a target node and boosting nodes. The spammer aims to hike the ranking of the target node through building the complete structure. The connected boost nodes are controlled by the spammers to influence the page rank of the target Web page. The boosting nodes are inherited by the creator of the target pages or by spammers interested to collaborate with the owner financially or otherwise. The primary purpose of the boosting nodes is to improve the ranking of the target page having no specific importance of its own. Huge number of boosting nodes are used by the spammers to initiate high ranking for the target as they usually have very small page rank. The paper proposed spam detection methodology that identifies the nodes whose PageRank scores are majorly boosted through spam links. The methodology is based on evaluating the spam mass of the nodes which estimates the comparative PageRank contribution of the linked spam pages. Combining the traditional PageRank and the other with biased random leap to some acquainted good nodes spam mass estimate is computed.

D. TrustRank

[5]TrustRank is semi-automatic methodology to segregate good trustworthy from spam. Initially, a small set of seed pages are selected to manually evaluate by a professional to identify the trustworthy seed pages. Then other probable good pages are identified using web link structure. As discussed, the method involves subjective human evaluation to infer a page as spam an approach, a binary oracle (prediction) is formulated for analysing if a page is spam. The function O over all the pages is expresses as,

$$p \in V: O(p) = \begin{cases} 0, & \text{if p is bad} \\ 1, & \text{if p is good} \end{cases}$$

The ideal trust property is defined as,

$$T(p) = \mathrm{Prob}[O(p) = 1]$$

For illustration set of 100 pages is considered and trust score assigned to each page as 0.7.

Suppose using the oracle function all the pages in the set is evaluated. If T estimates properly then the oracle score will be 1 for 70 pages and 0 for the rest 30 pages.

It is difficult to formulate T that measures accurately the likelihood of the page being good. However, it facilitates to order the pages with respect to the probability of the page being good resulting in ordered search results.

Suppose there are two pages p and q respectively. The page p with low trust score than q for example hints at page p being less likely to be good. Thus, the important property of the trust function is the ordered trust property. This can be expressed as,

$$T(p) < T(q) \Leftrightarrow \mathrm{Prob}[O(p) = 1] < \mathrm{Prob}[O(q) = 1],$$

$$T(p) = T(q) \Leftrightarrow \mathrm{Prob}[O(p) = 1] = \mathrm{Prob}[O(q) = 1].$$

Further, through introduction of a threshold value σ the compulsions for T can be relaxed. Threshold trust property is thus expressed as,

$$T(p) > \delta \Leftrightarrow O(p) = 1$$

With the exponential growth volume and value of the web search engine plays a vital role providing diverse users to search information of their interest. However, the search engines are majorly threatened by perilous web spams leading to unbiased search and page ranking. The TrustRank methodology has suceesfully addressed this major issue in detecting the spam links through filtering the index separately or combined with the PageRank providing running time assessment of diverse memory.

E. PageRank Computation

The paper [6]discusses computation of PageRank on machines with constrained main memory for huge subgraphs

of web providing running time assessment of diverse memory configuration across millions of pages archived in Standford WebBase. Furthermore, based on the page ordering several analysing techniques for PageRank convergence is explained. The convergence results are a driving agent in presence and absence of search queries to determine the iteration counts required to assign complete PageRank.

Algorithms exploiting the web link structure are emerging as powerful device for providing relevant search queries results. Despite the simplicity of the algorithm scaling the operational implementation on large scale web subgraphs demands for accurate data arrangements. It has been researched that PageRank algorithms can be executed on modest rigged computers and a single precision vectors of Rank can yield accurate computation. The convergence algorithm can be analysed in several ways depending on the induced ordering on the pages through the single precision Rank vector. The paper has stated from the experimented output convergence rate the useful Rank vector can be computed on a modest rigged computer approximately in one hour which demonstrates the computational task of PageRank from the perspective of client-side feasibility.

F. Topic-sensitive PageRank

Single precision Rank vector do not have dependency on specific search queries in identifies the relative importance of the Web pages referring to the Web link structure. An improvement over the single precision Rank vector is proposed in [7]to result in further accurate search results. It is proposed to calculate a set of Rank vectors introducing a bias through using a set of similar topics. This captures the relative importance with high accuracy having particular reference topic. The topic-sensitive PageRank scores are computed for instances of typical search queries for the pages that satisfies the query utilizing the field of the query keywords. Whereas, for instances of context search it is computed using the field of the context where the query aroused. The linear combination of the of the above-mentioned biased Rank vectors to yield importance score based on the context during the time of query for the page provides improved accurate rankings.

Implementing the traditional PageRank algorithm, the importance scores are initially calculated offline. A set of importance score is computed for each page with respect to different topics. Depending on the query topic the importance scores are combined resulting in a compound PageRank Score of the pages coordinating to the queries. Combined with the IR-based scoring techniques the former generates the final rank for the pages parallel to the queries. However, IR scores for commercial search are eliminated in the research as the score function for the same was unknown. Irrespective of the IR-based score factor overall search ranking will improve with the improved PageRank precision.

Another area of research to build the best set containing basis topics is left unturned. The cost of the simple methodology is directly proportional to the basis topic number.

G. Authoritive sources

[8]The information about the content of the hyperlinked environment can be sourced from structure of the network considering the effective understanding resources. Algorithm based several tools are built to extract the information of the environment from the structure. Effectiveness is demonstrated in varied context on World Wide Web. The primary issue is of refined topic search through the authoritative source of information on the topics. The algorithm is based on the relationship between set of authoritative and hub pages joined through the link structure. This algorithm is also termed as HITS (Hyperlink-induced topic search). Unlike PageRank the purpose of the algorithm is to rank the result of the query only executing along with the search query processing. Important pages are viewed by HITS form two important perspective as discussed below:

a) The pages termed as authorities provides details of a topic and hence considered valuable.

b) The hub pages are valuable as they provide direction to find information about the topic.

Accordingly, it is proposed to assign two scores to every Web page for accessing the degree of good hub and good authority respectively. Hubbiness is estimated through adding successor's authority and authority is estimated by adding predecessors' hubbiness. However, to limit the boundary the elements of the respective vectors are scaled in a way that the largest element limits to 1 or the sum limits to 1.

II. DATA MAP

Data Source

The data used the calculation of improved PageRank is sourced from Common Crawl specifically archived crawl for February 2020. Over the past 7 years petabytes of data is collected and stored here. Raw HTML data, metadata and text extractions are the primary content of the source. The Common Crawl datasets are located in public AWS public S3 buckets. The February 2020 archive stores 2.6 billion web pages crawled in the interval of 16^{th} – 29^{th} February 2020.

Data Format

At present the Common crawl archive stores data format using Web ARChive (WARC). Prior to this in ARC file format it was stored. The advantage of the former mentioned format is that it facilitates storage and processing of hundreds of free terabytes of web pages archives more efficiently. However, in this project WAT response format is used.

This file format stores important metadata which is calculated corresponding to the three kinds of records: request, metadata and response in the WARC format. For HTML crawled information the calculated metadata contains the returned HTTP headers and the listed links on the page. As this is stored in JASON to potentially reduce the size the irrelevant whitespaces are stripped resulting in unreadable format.

In this project WAT metadata is used for the creation of linkgraph. 'WAT-Target-URI' and HTML-Metadata Links are extracted to get the website url and the outlinks of that website.

Data Access

The data is accessed from the S3 buckets using 'warcio' library using which WARC files can be read as a stream of records.

III. METHODOLOGY

Google explains page rank as the links between pages as vote. The significance of a page is determined through counting the relevant link votes. The scores combined with other determining factors asserts the Rank of the pages. The page Rank for say page can be expressed as:

$$\text{PR(A)} = \beta\left(\frac{PR(T1)}{C(T1)} + \cdots + \frac{PR(Tn)}{C(Tn)}\right) + (1-\beta) \quad (1)$$

Where T_i, (i=1…n) are the number of pages citing the page A and C(A) are the counts of outgoing links of page A. Here the parameter β is the probability of selecting random outgoing link (probability is lost) and (1- β) as the probability of random jump (Probability is shared). The model is termed as PageRank Graph hopping Random Surfer which is based on random walk. A random surfer will start at a random node and selects a random out link with the probability of β or jumps to land at a random node with a probability of (1- β). Additionally, the PageRank values add up to 1 as it is a probability distribution over the Web pages.

PR(A) can be computed through iterative simple algorithm which associates with the principal eigen vector the Web's normalised link matrix.

The pages with no out links are termed as Dead end. A page loses its PageRank eventually if it leads to the dead end as it has no links to any other pages to distribute the scores. The pages that have links pointing to itself is called spider traps. This will lead to contrate the PageRank to itself and not distribute to the neighbours.

Improved PageRank

To address the problem of dead ends and spider traps the former equation is modified as follows:

$$\text{PR(A)} = \beta\left(\frac{PR(T1)}{C(T1)} + \cdots + \frac{PR(Tn)}{C(Tn)}\right) + (1-\beta)/N \quad (2)$$

The PR(A) however remains random column matrix with positive entries. Say from page A the surfer moves to the neighbourhood following the outgoing link. There is a small probability that the suffer discards the present page and randomly selects a different page in the web to teleport there. As the surfer can teleport to any of the webpage there is 1/n chance for each page to be selected. Now, the dead-end node has $(1-\beta)/N$ probability to move to another node.

A. Problem of Leaked PageRange

As discussed early the dangling node or the dead end loses the probability. Three computational tasks to address the issue along with respective challenges is discussed below.

i. To obtain a sum of probabilities of all the nodes equals to 1 the PR is renormalized. However, for high incoming nodes PR will be overestimated.

ii. To connect the sink node with every node of the link graph to distribute the PageRank equally. Generally, the link graph is sparse. Thus, if the sink node is connected to every node it will result in denser graph which eventually makes the computation expensive and not feasible.

iii. To prevent the probability loss the PageRank is formulated as below:

$$PR(A) = \frac{1-\beta}{N} + \beta\sum_{B\to A}\frac{PR(B)}{C(B)} + \beta\sum_{Z\to\phi}PR(Z)/N \quad (3)$$

Instead of connecting the sinking node with every node the probability of it is summed up and equally distributed among the other nodes intuitively.

B. Modified Trust Rank

Topic-sensitive PageRank is termed as Trust Rank. Here, topic is referred to set of trustworthy pages. The basic assumption backing the concept is that a spam page is probable to link to trustworthy pages, but the reverse is unlikely.

The idea is to introduce a factor $\tau < 1-\beta$ which will be used for the calculation of PageRank for the trustworthy pages having extensions like .edu, .gov etc. This will restrict the distribution of PageRank of trustworthy pages however maximizing its own probability. It is expected this will reduce the problem of spam farm.

$$PR(A)\frac{1-\beta-\tau}{N} + (\beta+\tau)\sum_{B\to A}\frac{PR(B)}{C(B)} + (\beta-\tau)\sum_{Z\to\phi}PR(Z)/N \quad (4)$$

C. Power Iteration

The PageRank algorithm is based on the Power iteration method of linear algebra. This facilitates to calculate eigenvector corresponding to the largest eigen value of the transition matrix. Below is the detailed explanation.

For a web graph with N number of nodes the pages are represented by the nodes and the websites are the hyperlinks. The ranking vector r is initialized as,

$r^0 = [1/N,\ldots..,1/N]^T$ where N is the number of web pages. The recursive equation is iterated as follows,

$r^{t+1} = M.\ r^t$ where M being the stochastic column matrix. The iteration stops at $|r^{t+1} - r^t| < \varepsilon$. The r can be also described as the stationary distribution for the random surfer movement. The stationary distribution is unique for certain satisfying conditions on the graph. This in turn means that irrespective of the initialization value of the ranking vector the power iteration converges to r.

IV. IMPLEMENTATION AND ACHITECTURE

The section discusses the implementation of improved PageRank algorithm over distributed Hadoop MapReduce framework. The classical PageRank algorithm using MapReduce can be decomposed into multi step task.

PageRank Using MapReduce

1 **procedure** Mapper⟨y, {x1 ….. xn}⟩ *#node +outlinks*
2 **for** j=1…n: **do: emit** $\langle \boldsymbol{x_j}, \frac{\boldsymbol{PR(y)}}{\boldsymbol{out(y)}}\rangle$
3 **end for**
4 **emit** ⟨y, {x1 ….. xn}⟩
5 **end procedure**
6 **procedure** Reducer$\langle \boldsymbol{x}, \{\frac{\boldsymbol{PR(y1)}}{\boldsymbol{out(y1)}}, \ldots, \frac{\boldsymbol{PR(yn)}}{\boldsymbol{out(yn)}}, \{\boldsymbol{x1}, \ldots \boldsymbol{xn}\}\}\rangle$
7 $PR(x) \leftarrow \frac{1-\beta}{N} + \beta\sum_{B\to A}\frac{PR(y)}{C(y)}$
8 **for** j=1…n: **do: emit** $\langle \boldsymbol{x_j}, \frac{\boldsymbol{PR(y)}}{\boldsymbol{out(y)}}\rangle$

```
9                   end for
10                  emit⟨x, {x1 ….. xn}⟩
11 end procedure
```

To use the above algorithm to calculate PageRank on common-crawled data it is sub divided into five tasks. First, data parsing task to extract website name and out links to other websites as key value pair followed by calculation of graph size. Then initial PageRank is distributed among all the nodes. To tackle the problem of leaked PageRank the cumulative probability of all the dangling nodes are computed followed by final PageRank calculation. The last two tasks are iterated until the PageRank converges.

Linkgraph

```
1   class MRCalculate_Link_Graph()
2       procedure mapper(Path)
3                   res ← response(url)
4                   page_domain ← parse(res)
5                   links ← parse(res)
6                   emit page_domain, array(links)
7       procedure Reduce(domain, links)
8                   links ← array(links)
9                   domain,{'id':domain,'state': 0,'outgoing':
    links}
10  end class
```

The code is implemented to generate the link graph on which the PageRAnk calculations is performed. The program accesses the common crawl data to parse and extract the host names of websites along with the out links.

Mapper: Accessing the s3 buckets extracts the WAT files from the common crawl repository to obtain the host names and the out links. The key value pair is expressed as,

⟨page_domain, array(links)⟩

Reducer: Here the key value pair expressed in mapper is transformed into another key value pair as expressed below,

⟨domain, {'id': domain, 'state': 0, 'outgoing': links}⟩

Size of Linkgraph

```
1   class CountGraphNodes()
2       procedure mapper(website,node)
3                   emit '_', website
4                   for all outgoing ∈ node['outgoing'] do:
5                           emit '_', outgoing
6                   end for
7       end procedure
8       procedure reducer(_, websites)
9                   emit 'count', length(websites)
10      end procedure
11  end class
```

For the calculation of PageRank algorithm total number of node present in the linkgraph need to be calculated initially. Above algorithm performs the task in the implementation.

Mapper: In this section all the host links and the outgoing links for a website is emitted consecutively. Key value pairs are given as, ⟨'_', website⟩ and ⟨'_', outlinks⟩ respectively.

Reducer: Here the emitted values are counted one after another and the resulting sum is emitted as a key value pair as ⟨'count', length(websites)⟩.

Distribute Initial PageRank

```
1   class Init_PageRank()
2       procedure configure_args()
3                   cmd_args("—graphsize")
4       end procedure
5       procedure mapper(website,node)
6                   node['state'] ← 1.0/--graphsize
7                   emit website, node
8       end procedure
9   end class
```

The initial step of PageRank calculation involves distribution of equal PageRAnk to all the nodes i.e. for say N nodes each is assigned with 1/N.

Mapper: Each state value which represents the PageRank value of a website is calculated and initialized.

Dangling Node PageRank Sum

```
1   class DanglingNodeJob()
2       procedure mapper(website, node)
3                   if length(node['outgoing']) = = 0 Then
4                           emit '_' , node['state']
5                   else:
6                           emit '_' , '0'
7                   end if
8       end procedure
9       procedure reducer( _, states)
10                  emit '_' , sum(states)
11      end procedure
12  end class
```

To address the problem of leaked probability all the PageRank value of the dangling nodes are summed to distribute it among the other nodes equally.

Mapper: Here the counts of the outgoing links for a website is checked. If the count is zero meaning a dangling node the PageRank of those websites are emitted as key value pair.

⟨'_' , node['state']⟩

Reducer: All the emitted PageRanks of the dangling nodes are summed to find the lost PageRank which will be distributed while calculating the PageRank.

⟨'_' , sum(state)⟩

PagrRank Calculation

```
1 class PageRank()
2       procedure configure_args()
3                   --graph-size ← 0
4                   --dangling-node-pr ← 0.0
5                   --damping-factor ← 0.85
6                   --Trust-factor ← 0.00
7                   --Trust-domain ← ['edu', 'gov']
8       end procedure
9       procedure mapper(website, node)
10                  emit website, ('node', node)
11                  if--Trust-domain in website.split('.') Then
12                          --Trust-factor = 0.10
13                  end if
14                  for all outgoing ∈ node['outgoing']:
15                          msg ← node ['state'] /
                        length(node['outgoing' ]
16                      emit outgoing, ('msg', msg)
17                  end for
```

```
18        end procedure
19        procedure reducer(website,data)
20                node ← None
21                for all msg_type, msg_val ∈ data Then
22                        if msg_type = = 'node':
23                                node ←  msg_val
24                        else if msg_type = = 'msg':
25                                msgs.append(msg_val)
26                        end if
27                if node != None Then
28                        node['state'] ← (--damping-factor +
--Trust-factor) * sum(msgs) +
                          (--damping-factor + --Trust-factor
                          *  --dangling-node-pr)  /  --
                          size_of_web  +  (1  -  --damping-
                          factor – node['trust'])/ --size-of-
                          web
29                end if
30        end procedure
31    end class
```

The above-mentioned final algorithm takes inputs from the prior results to calculate the improved PageRank value of all the nodes.

Mapper: In this section of the code the domain of the website is evaluated to calculate the Trust factor. This is especially checked for the educational and government websites having .edu and .gov domain respectively.

Reducer: The improved PageRank value is calculated using the equation (4). For the above calculation the dangling node probability is taken from the output calculated through the algorithm of dangling node PageRank sum and the graph size from size of linkgarph algorithm respectively.

Cumulative PageRank Difference

```
1     class DanglingNodeJob()
2         procedure mapper(website, node)
3                 if length(node['outgoing']) = = 0 Then
4                         emit '_' , node['state']
5                 else:
6                         emit '_' , '0'
7                 end if
8         end procedure
9         procedure reducer( _, states)
10                emit '_' , sum(states)
11        end procedure
12    end class
```

The above algorithm calculates the cumulative difference of all the PageRanks i.e., the PageRank difference between two consecutive states of a particular Web page. It is expressed as, $\sum_j |PR(A_j)_{t-1} - PR(A_j)_t|$

Job Runner

The consecutive algorithms Dangling Node PageRank Sum, PagrRank Calculation, Cumulative PageRank Difference respectively are iterated until convergence. The convergence of PageRank is said to be accomplished by constraining the Cumulative PageRank Difference with ε which is a small number greater than 0. The process is based on the theory of Power iteration implementation using a job runner to run the consecutive task in an iterative manner. The results of the previous algorithms can be used to improve the generic Power method-based PageRank algorithm, for the approximate computation of the PageRank derivatives till the convergence of the algorithm.

Hadoop Achitecture

The Hadoop framework was installed in a cluster on distributed environment. The cluster consisted of one master node and two data nodes using the OpenStack private cloud service of National College of Ireland.

V. RESULTS

This section concentrates on the implementation and experiments of improved PageRank algorithm in Hadoop distributed environment, Using Hadoop 3.2.1 with Python3 the improved model is implemented where each task is split into Map and Reduced task. The system input is WAT paths of the AWS S3 buckets of Common crawl data of February 2020. The Hadoop cluster was installed on cloud instances with configuration 2V CPUs, RAM 4GB and Storage 40GB for all master nodes and data nodes.

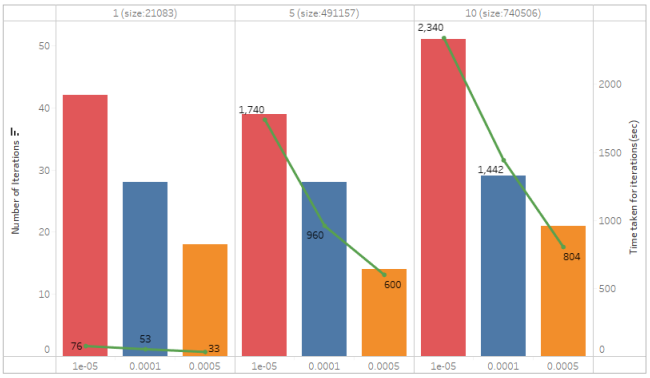

Fig:1

The graph in fig:1 is plotted with x-axis as the convergence factor and the y-axis represents dual axis for number of iterations and time taken for convergence. The bars are coloured with respect to the convergence factor ε. The results demonstrated in the graph explains that with decrease in convergence factor the number of iteration required for the convergence approximately doubles at each test. Number of iterations is directly aligned with the time taken for convergence. The above results are tested against the ε values: 0.0001, 0.0005, 0.00001 respectively.

VI. CONCLUSION

The improved PageRank algorithm proposed in this paper is based on the convergence property of Power Iteration method. Using the Hadoop architecture, it was implemented over the distributed environment. The algorithm is sub divided into six tasks: Linkgraph creation, size calculation of Linkgraph, initializing Linkgraph, sum of dangling node PageRank value, PageRank calculation and total page rank difference following an iteration. The output result of the implementation has showed that with the decrease in the convergence factor ε the number of iterations and the time taken to convergence approximately doubles at each test.

ACKNOWLEDGMENT

I take this opportunity to convey my heartfelt gratitude to the professor of Scalable Systems Programming module Dr. Horacio Gonz´alez–V´elez who has constantly extended required support throughout the project work.

REFERENCES

[1] A. Bhawiyuga and A. P. Kirana, “Implementation of page rank algorithm in Hadoop MapReduce framework,” *Proceeding - 2016 Int. Semin. Intell. Technol. Its Appl. ISITIA 2016 Recent Trends Intell. Comput. Technol. Sustain. Energy*, pp. 231–236, 2017, doi: 10.1109/ISITIA.2016.7828663.

[2] S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine BT - Computer Networks and ISDN Systems,” *Comput. Networks ISDN Syst.*, vol. 30, no. 1–7, pp. 107–117, 1998, doi: 10.1016/S0169-7552(98)00110-X.

[3] A. Broder *et al.*, “Graph structure in the Web,” *Struct. Dyn. Networks*, vol. 9781400841356, no. 1, pp. 183–194, 2011, doi: 10.1145/2615569.2615674.

[4] Z. Gyongyi, P. Berkhin, H. Garcia-Molina, and J. Pedersen, “Link spam detection based on mass estimation,” *VLDB 2006 - Proc. 32nd Int. Conf. Very Large Data Bases*, pp. 439–450, 2006.

[5] H. Garcia-molina and J. Pedersen, “Combating Web Spam with TrustRank,” no. 1973, pp. 2004–2004, 2004.

[6] T. H. Haveliwala, “Efficient Computation of PageRank,” *Stanford Univ.*, pp. 1–15, 1999, doi: 10.1145/971701.50239.

[7] T. H. Haveliwala, “Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search,” *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 4, pp. 784–796, 2003, doi: 10.1109/TKDE.2003.1208999.

[8] J. M. Kleinberg, “Authoritative sources in a hyperlinked environment,” *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999, doi: 10.1145/324133.324140.