Page Rank using Hadoop Cluster

Karan Mahendra Singh
18192726
Scalable Systems Programming
National College of Ireland
Dublin, Ireland
karanmsingh10@gmail.com

***Abstract*—With too much data on the internet, it is essential that right pages show up when users enter search queries. Therefore, search engines strive to bring up only relevant pages in search results by ranking pages according to their importance. This subsequently leads to users' preference of using search engines. PageRank algorithm is used for ranking webpages and is calculated from the graph of the internet. This includes in-links and out-links to and from a webpage. Further, increasing amounts of organizational data has led to a demand for robust systems capable of their storage, extraction, transformation, and processing. Therefore, newer systems for analyzing and processing such large data by utilizing concepts such as parallelization, distributed storage and scalability were developed. These systems helped reduce computational time and enabled efficient usage of computing power. Hadoop and MapReduce are commonly used for processing of such humongous data. This project employs a distributed Hadoop cluster for calculating page ranks of webpages included in a Common Crawl dataset. Furthermore, this paper brings an account of embarrassingly parallel data stream processing algorithms which use MapReduce and/or distrusted file systems and some of them are applied in companies like Google and Yahoo.**

***Index Terms*—Hadoop, MapReduce, PageRank**

I. Introduction

The size of the world wide web kept growing as people became more dependent on it. Businesses, news, and other information moved to the internet for a larger reach to the viewers. Users relied on the internet too, for searching and gathering information on various topics. Therefore, a way of listing relevant pages on typing a search query was required. This would improve the search experience of users of the internet.

Consequently, search engines devised a way of finding webpages that were relevant to the search. In that, they looked for the exact words used in the search query and matched them with titles of webpages. This established the page to be more relevant than those which contained them in normal texts. Spammers saw opportunity in this system and copied words which people searched for, in their webpages. This made their pages appear important to the crawlers and it was listed high in rank in searches for those words [4].

Furthermore, numerous webpages presented information over a wide array of topics. Unlike legitimate academic sources which are ethically presented and reviewed before publishing, these unauthentic sources propagated material without any quality control. With a lot of businesses moving to the internet, in the quest for more profit, any strategy which can be tampered with copying of content, is vulnerable to forgery [1]. Methods used for tricking search engines into thinking differently of a page than it is, is called term spam [4]. In trying to prevent such unscrupulous acts, [1] proposed a novel technique called "PageRank" for calculating the rank of a page based on the graph of the web. Every page has several forward and backward links. The more the links, the more "important" it becomes. Further, even if a page has a single link from a highly important page, it becomes substantially more important than those having multiple links from other unimportant pages. This algorithm establishes importance of a page with the link structure of the web. The higher the number of back links a page has, the higher its importance. This is considered a vital element in building a search engine [5].

[3] Google uses PageRank to determine the importance and consequently, its rank. [2] it is important to arrange the large sub-graph data of the web for efficiently computing PageRank. If the above is done, it may be calculated easily on systems with limited resources.

Web mining is used to study users' behavior and the content on the web. It involves mining of the content, usage, and structure of the web. It further discovers the relationship among webpages by studying the link structure of the hyperlinks between them such as inlinks and outlinks. Furthermore, it keeps track of user profiles recorded on logfiles and the kind of knowledge in the overall content of the web [6].

This paper describes the application of Hadoop MapReduce for calculating PageRanks of hyperlinks available in a dataset obtained from Common Crawl. A single node execution was compared with a Hadoop distributed cluster-based execution.

Thereafter, a background of scalability and algorithms using MapReduce environments is presented in this work. With rising amounts of data in organizations, they required systems to store and process them. Therefore, newer systems were developed that enabled analysis of large-sized datasets. These systems were based on concepts of parallelization, scalability, and cluster computing to not only enable their processing, but also do it within shorter time frames. They work parallelly, with their work often divided amongst a cluster of nodes. [12] each node gets its own chunk of data to process and performs independently without requiring to communicate with other nodes, known as *embarrassingly parallel.*

This paper is divided into sections as follows: section II gives a background study on scalability and algorithms. Section III describes the dataset used in the study. Section

IV states the methodology followed, and section V provides the implementation of MapReduce and the scalability measure applied. The findings of this study are reported in section VI under the title Results and finally, section VII comprises of the conclusion and discussion for the future work.

II. Background on Scalability and Algorithms

This section briefly describes parallel systems followed by an in-depth account of parallel algorithms implemented using MapReduce.

Big-data analysis processes large volumes of data as quickly as possible. Such bigdata can be processed at high speeds with the use of parallelism. Systems employ parallelism through cluster computing. Each device capable of computing in the cluster, is known as a node. The advent of cloud computing made it easier to deploy such systems as the required cost was dramatically reduced. A distributed file system is used for replication of data as a measure against data loss due to the failure of a node. Essentially, archival data, which is huge and rarely updated, is stored in these systems. They further assume that some node somewhere could fail to work and have measures to overcome such scenarios [1].

MapReduce is used with many systems such as the internal implementation of Google and Hadoop. Hadoop is often used with the HDFS file system, provided by Apache. The implementation of MapReduce is hardware fault tolerant and can be used for performing parallel computations on large scale data. The user process forms a master and some worker processes which execute two functions: Map and Reduce are used to define the working of the entire algorithm. The master process divides work among workers and keeps track of them. Map tasks receive chunks of data from the distributed file system, to perform operations and yield outputs in the form of key-value pairs, predefined by the programmer. A master controller takes all outputs of all the Map tasks, sorts them according to their keys and divides them among the Reduce tasks. The Reducer tasks combine all values of a key in a way that was again predefined by the programmer. Outputs from all Reduce tasks are merged into a file as the final output of the entire task. The method is prepared to face the failure of one of these tasks. Whenever a Map worker node fails, the master node discovers this soon enough as it keeps pinging all worker nodes. All tasks assigned to this worker are repeated as the Map output now becomes unavailable to the Reduce workers. The master allots this work to another worker when it becomes free and notifies the Reduce worker about the change of location of its input. However, the entire job must be restarted if the master node fails. Some other classes of operations performed using MapReduce are as follows:

A. Matrix-Vector Multiplication

Let there be a $n \times n$ matrix m and a vector v of the same length n. MapReduce is used when the value of n is substantially large. Further, the above vector is stored in the main memory of a compute node and made available to all mapping tasks. Each map function works on its share of the matrix elements to produce a key-value pair $(i, m_{ij}v_j)$ output. All terms contributing to the formation of the element xi in the matrix-vector product will have the key, i. Finally, the reducer aggregates all values of the key i.

B. Vector v cannot fit in main memory

The main memory of a compute node might not fit the vector at times. In such cases, as against Matrix-Vector multiplication, this method requires the matrix to be split into vertical strips of same width and the vector into horizontal strips of identical heights, thereby fitting the chunks of vector fit into the main memory. Thus, map tasks are provided corresponding strips of the matrix and vector. Further calculations are performed as in the above method. In applications such as PageRank, a better approach of dividing the matrix into square blocks instead of strips, is used.

C. Relational-Algebra operations

The possible applications of MapReduce in relational-algebra operations are developed by assessing typical operations on relations. These operations include selection, projection, natural join, grouping and aggregation and union intersection and difference. They may be performed by applying MapReduce.

D. Computing selections

If a tuple t in a relation R satisfies a condition C, the mapper function creates a key value pair (t,t) and nothing if it does not. The reducer function is the identity value and throws the key value as its output. This operation does not fully use MapReduce as it may be performed in only either one of the methods.

E. Projections

The map function creates a set of tuples t' with key-value as (t', t') from relation R such that it does not contain attributes apart from those mentioned in S. The reduce function in this method eradicates any duplicates created in the process and passes only one pair as output. This is different from other approaches seen so far where it is used to just aggregate all key-value pair outputs obtained from the map functions.

F. Union, intersection, and difference

While performing unions, mappers are allotted chunks either from relation R or S and turned into key-value (t,t) pairs. The reducer removes any duplicates before presenting the (t,t) output. The same map function is applied to compute intersections. However, the reducer generates a tuple only if it is cited in both relations. A slightly modified mapper is used for calculating difference $(R - S)$. It notifies which relation whether R or S, does the tuple come from. The reducer produces (t,t) only when the value relates to R. Mappers used in these operations do no compute much and just form key-value pairs while reducers work substantially more.

G. Natural Join

If we have two relations *R(A,B)* and *S(B,C)*, the mapper generates key-value pairs of both relations as: *(b, (R, a))* and *(b, (S, c))*. Later, the reducer forms all possible combinations of key-value pairs of the matched key pairs from both relations. The tuple therefore has values from R, key values, and values from S. The mappers used here are alike those used in finding difference, in that they pass the relation name to which the key-value pair belongs. Whereas the reducers used are entirely different.

H. Grouping and aggregation

If for a relation R, we have an attribute *(A)* for grouping, one *(B)* for aggregation and one *(C)* for none, the mapping function produces a key-value pair consisting of *(a,b)*. The reducer then matches all similar a values together to form a group and depending on the type of aggregation, for instance sum or max-value, performs the required operations on the list of b values matching the group key. In contrast with other algorithms, this method could allow the mapper to pass multiple values when there were a greater number of grouping attributes. Subsequently, the reducer performs aggregations on the lists of corresponding attribute values received.

I. Matrix multiplication

If there are two matrices *A and B*, it is imperative that the number of columns of A is equal to the number of rows of B. The matrix may be represented like a relation with three attributes *A (I, J, V)*. The tuples would be *A (i, j, m_{ij})* and likewise for *B: B (j, k, n_{jk})*. The multiplication MN can be simplified as a natural join followed by grouping and aggregation. The first map function passes a key-value pair of *(j, (A, i, a_{ij}))* for an element a_{ij}. The same is done for elements b_{jk}. The reducer takes the values at the above key-value pairs from both matrices and calculates their product, $a_{ij}n_{jk}$. Subsequently, another pass of MapReduce functions is applied where the mapper identifies the values to be summed and the reducer performs their summation forming *((i, k), v)*. v forms an element on the i^{th} row and k^{th} column of a newly formed matrix.

J. One step matrix multiplication

It is possible to perform the above operations with only one set of MapReduce functions by putting more load on each of them. The map function generates pairs elements *((i, k), (A, j, a_{ij}))* and *((i, k), (B, j, n_{jk}))* from each matrix require to compute an element in the resultant matrix. The matrix A and B are represented with a bit, as in the above operation. The pairs with same j values are linked, their third components which stand for the elements in respective matrices are multiplied and summed. The result is paired with the *(i, k)* as the output. Thus, the reduce function in this method performs more complex operations than seen in the previous method. While the mapper used here and the first mapper of are quite similar.

K. Other systems

A few other systems are capable of efficiently handling data streams with MapReduce or distributed file systems. These systems are described here.

A few other systems are capable of efficiently handling data streams with MapReduce or distributed file systems. These systems are described here.

[7] Apache Flink is a distributed architecture used for processing bounded and unbounded data streams. For bounded streams, certain methods and data structures are employed, while unbounded streams are processed using system states and controlled time. It can run at any scale using in-memory, to provide faster computation.

[8] Bigtable provides distributed storage which may be employed with MapReduce to perform large scale computations parallelly and achieve low latencies. A set of functions enables using Bigtable as input or output. Big table systems share machines among processes from multiple applications. Such high-performance systems are used at Google. It uses one library, many tablet servers and one master. It stores tables containing tablets of data-rows.

[9] Pnuts is again largely parallel, uses distributed storage with hashed tables and employs a message system for memory replicas. It was intended to provide data to web applications while carrying out various data operations with minimum latency. However, they are suited for more transactional operations of data with minimal latencies, than complex ones such as joins, group by and so on. These systems are used at Yahoo.

[10] Hive is based on Hadoop and enables accessing data through SQL. Data can be accessed using MapReduce, Spark or Tez. It can store data belonging to many formats with the help of connectors and is best for processing archival data. Hive cannot be used for transactional systems. It provides the entire process of warehousing which involves ETL and data analysis.

[11] Pig provides ad-hoc analysis of large datasets. It provides more high-level control than MapReduce and is more procedure-oriented than SQL. MapReduce has some drawbacks such as the two-step dataflow, lesser code reusability and restricted optimization. Pig is executed with Pig Latin and can be run over Hadoop. This system is used at Yahoo for data analysis tasks as it reduces execution time.

III. Data Map

This section describes the data used for the study.

The dataset used in this project was sourced from Common Crawl. It sources crawled data of the internet and makes it publicly available to promote research and analysis by technology enthusiasts.

Crawl data of May/June 2020 was used [13]. The website stores crawl archive data in formats like WARC, WAT and WET. The WAT file was used here.

IV. Methodology

This section describes the method used in this project for calculating page rank from the above dataset.

The entire web can be compared to a directed graph where every web page is a node. These pages are connected with each other through in-links and out-links which form links connecting the nodes in a graph. The direction of a link indicates whether it is an in-link or out-link of a node or web page. Fig. 1 shows the link structure of the web.

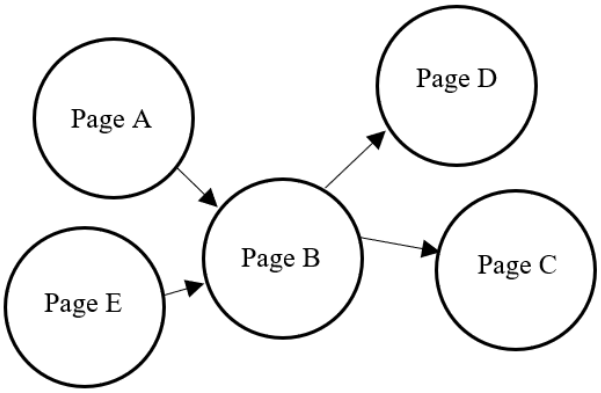

Fig. 1. Internet as a Directed Graph

It is seen from Fig. 1, that page B has two out-links to pages C and D and two in-links from pages A and E. The PageRank algorithm establishes a page more important if it has more in-links. Moreover, in-links from other important pages makes the page more important. Thus, from Fig. 1, Pages D and C would be more important than pages A and E, as they have in links from an important page B.

Many other elements were also calculated while calculating the page rank of the links mentioned in the dataset. They are as follows:

1) Link graph
   First, the dataset of [13] was used to form link graphs. The output from this formed the base for the most of other elements.
2) Dangling links
   From the above output, the dataset was checked for any dangling links which are links to a page with no out-links.
3) Graph size
   Graph size was calculated with the link graph formed previously.
4) Initial page rank
   This step again used the link graph to calculate initial page ranks.
5) Page rank
   Finally, the initial page ranks calculated in the above procedure was used to compute final page ranks.

V. Implementation and Architecture

The implementation and architecture of the system is explained as follows.

A Distributed Hadoop Cluster was used to calculate page ranks in this study. All instances were created through OpenStack and connected to form a cluster. The dataset was stored on Hadoop Distributed File System to avail all benefits such as data replication – for incase a node fails and a large storage – given the size of internet and in general, large archived organizational data. The cluster was formed to achieve scalability and consequently, a sped up computational time. Fig. 2 depicts a Hadoop cluster.

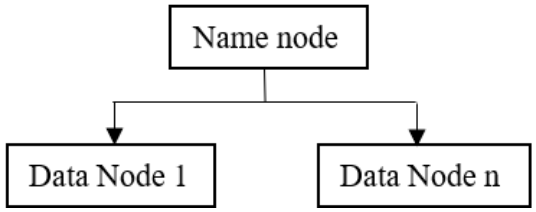

Fig. 2. Hadoop Cluster

Further, MapReduce was used to distribute computations amongst the nodes. The Mapper mapped memory chunks of links in the dataset, to each node and formed keys which was then used to sort the intermediate output into order. The Reducer then collected all data, matched the keys, and formed one aggregated output from the fragmented output of the mapper. Fig.3 shows the workflow of the system used in this project.

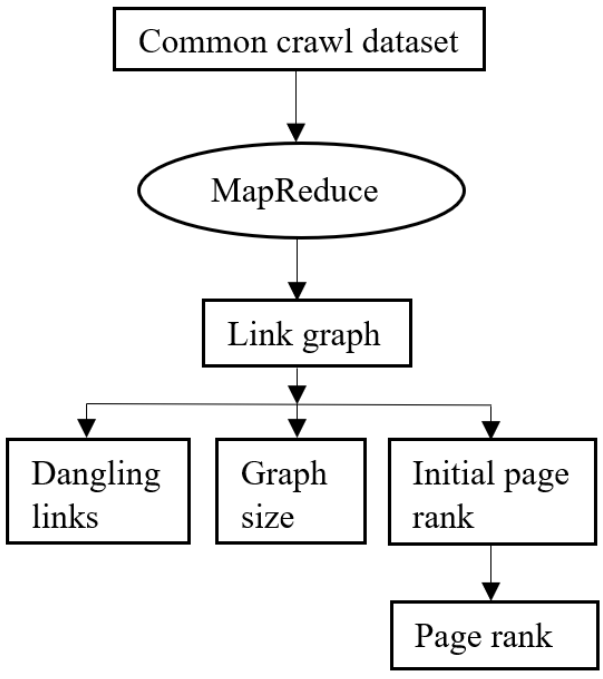

Fig. 3. Workflow

VI. Results

Following the above method, the dataset May/June 2020 dataset obtained from common crawl was used to create a

link graph. Subsequently, initial page ranks were calculated from this. Finally, the page ranks were computed from the initial page ranks. The entire code was written in python. It was further observed that achieving scalability by adding more nodes results in lesser execution time. Fig. 4 shows the final page ranks, where the value after "state" is the page rank value of that page.

Fig. 4. Page Rank

VII. Conclusions and Future Work

Thus, a Hadoop cluster was used to deploy the PageRank algorithm. MapReduce was used in conjunction with the HDFS to distribute the calculation task among two nodes. The aim of the study of page rank analysis using the common crawl dataset was thus met. Moreover, other data stream processing algorithms deployed using MapReduce and/or distributed file system were described in section II. In future, more nodes or other algorithms such as Pig, Spark or Hive could be used for calculating PageRank.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web. Technical Report," 1999 Stanford InfoLab.

[2] T. Haveliwala, "Efficient computation of PageRank," 1999 Stanford.

[3] S. Brin, and L. Page, "The anatomy of a large-scale hypertextual Web search engine," 1998 Computer Networks and ISDN Systems, 30(1-7), pp.107-117.

[4] J. Leskovec, A. Rajaraman, and J. D. Ullman, Mining of massive datasets. Cambridge: Cambridge University Press, 2020.

[5] Y. Zhang, L. Xiao and B. Fan, "The Research about Web Page Ranking Based on the A-PageRank and the Extended VSM," 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, Shandong, 2008, pp. 223-227, doi: 10.1109/FSKD.2008.267.

[6] W. Xing and A. Ghorbani, "Weighted PageRank algorithm," Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004., Fredericton, NB, Canada, 2004, pp. 305-314. doi: 10.1109/DNSR.2004.1344743

[7] Flink.apache.org, "Apache Flink: What is Apache Flink? — Architecture", 2020. [Online]. Available: https://flink.apache.org/flink-architecture.html. [Accessed: 18- Jul- 2020].

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable," ACM Transactions on Computer Systems, vol. 26, no. 2, pp. 1–26, 2008.

[9] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts," Proceedings of the VLDB Endowment, vol. 1, no. 2, pp. 1277–1288, 2008.

[10] Cwiki.apache.org, "Home - Apache Hive - Apache Software Foundation", 2020. [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/Home. [Accessed: 25- Jul- 2020].

[11] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08, 2008.

[12] W. Neiswanger, C. Wang, and E. Xing, "Asymptotically exact, embarrassingly parallel MCMC," arXiv preprint arXiv:1311.4780, (2013).

[13] S. Nagel, "Common Crawl," 10-Jun-2020. [Online]. Available: https://commoncrawl.org/2020/06/may-june-2020-crawl-archive-now-available/. [Accessed: 05-Aug-2020].