ANALYSIS OF PAGERANK ON WIKIPEDIA

by

ANIRUDH TADAKAMALA

B.Tech, Jawaharlal Nehru Technological University, 2012

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2014

Approved by:

Major Professor
Dr. Daniel Andresen

Abstract

With massive explosion of data in recent times and people depending more and more on search engines to get all kinds of information they want, it has becoming increasingly difficult for the search engines to produce most relevant data to the users. PageRank is one algorithm that has revolutionized the way search engines work. It was developed by Google`s Larry Page and Sergey Brin. It was developed by Google to rank websites and display them in order of ranking in its search engine results.

PageRank is a link analysis algorithm that assigns a weight to each document in a corpus and measures the relative importance within the corpus. The purpose of my project is to extract all the English Wikipedia data using MediaWiki API and JWPL(Java Wikipedia Library), build PageRank Algorithm and analyze its performance on the this data set. Since the data set is too big to run in a single node Hadoop cluster, the analysis is done in a high computation cluster called Beocat, provided by Kansas State University` Computing and Information Sciences Department.

Table of Contents

List of Figures

List of Tables

Acknowledgements

On the very outset, I would like to thank Dr. Doina Caragea for having faith in my abilities and giving me an opportunity to work on the current project.

I would like to thank to Dr. Daniel Andresen for being my mentor and for his constant guidance during the course of this project. I would also like to thank Dr. Torben Amtoft for being part of my committee, reviewing my work thoroughly and giving his valuable feedback.

I am very grateful to Dr. Gurdip Singh and Dr. Rodney Howell for providing me the opportunity to work as a Graduate Teaching Assistant. It helped me learn a great deal from this work experience.

I am grateful to my family and friends for all their love, encouragement and constant support.

Chapter 1 - Introduction

Motivation

Search Engines before PageRank [1]

One of the first search engines World Wide Web Worm (WWWW) was developed in the early 1990s. It claimed to have indexed 110,000 web pages in 1994 and from there on various search engines claimed indexing documents up to 100 million in number. The number of user queries had increased from 1500 per day in 1994 to 20 million per day in 1997 [1]. This shows how exponentially the number of internet users increased during that period.

The existing search engines did not employ algorithms that automated the process of information retrieval. This resulted in lot of challenges, but since the user did not know what was coming in the next few years, he was just satisfied with the results he was getting. This resulted in two major challenges for the search engines.

Scalability:

With the number of web users increasing in millions every day, the then technology went on to become slower and slower on responding to user requests. Scaling to such humongous population on their application did not present them an easy task. To some extend this challenge was overcome by increasing the hardware efficiency. However it was cost effective.

Quality:

The search engines depended on human to index millions of documents that contained possibly all the popular information the users might want. However, this led to a lot of useful, less popular topics go missing from the search engines` databases [1]. Hence users did not get most accurate results sometimes and no results at all for certain topics that are totally off beat.

These two and many other major challenges made it more and more difficult for the search engines to succeed in the web market place and they ultimately saw the need for a new technology that would address the abovementioned challenges.

After thorough research, two PhD students at Stanford University had developed an algorithm that has revolutionized the web search and made them owners of one of the most successful companies in the world.

Solution to the above problem: PageRank Algorithm

Sergey Brin and Lawrence Page published a paper titled "Anatomy of a Large-Scale Hyper textual Web Search Engine" in 1998 to describe a search engine they had developed called Google. In this paper they described PageRank algorithm that they have developed for Google.

Technological changes and the growth of Web were the two main driving factors in developing Google. So it was designed to scale efficiently to very large data sets. The data structures they used to develop this application made sure the data access was very fast. They stored index of the web pages in the space that was available to them and hoped this requirement for large amounts of space would eventually reduce.

Due to the tendency of people to only look at the first few results returned by the search engine, there was need for an algorithm that very precisely identified the most relevant documents for a query. In this context, two terms Precision is the number of relevant documents in the top ten results and Recall is the total number of relevant documents the system identifies. So high precision means more relevant information is accessed by users.

Another goal of Google was to make search engines as academic as they are commercial. Prior to Google, search engines mostly drifted towards commercialization and so the results of search engines mostly were business centric rather than academic.

To achieve the goal of producing high precision results, Google made use of the link structure on the web to calculate relative quality ranking of each web page. This algorithm is called **PageRank**.

Definition of PageRank[1]

PageRank is the objective measure of the citation (pointer to page, usually a hyperlink) importance of a page that corresponds well with the subjective importance of user`s interest

In what follows, I present the formula to calculate PageRank of a page A (PR(A)).

$$PR(A) = (1-d)/N + d\,(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))$$

T1,T2…Tn are pages that point to A, i.e., in-links of A. d is called Damping Factor which is set between 0 and 1. Generally it is set to 0.85. C(Ti) is the number of citations Ti has,i.e., the outlinks of Ti.

If a corpus has N documents T1,T2,…Ti.. , then the summation of all page ranks, PR(T1) + PR(T2) + PR(T3) + ….. = 1.

Thus the page ranks of all documents in a corpus, as defined by the algorithm form a probability distribution.

The basic idea behind this formula is that a page or a document distributes its rank equally to all its out-links. So each page gets a portion of its rank from each of its in-links. The maximum possible PageRank for any given page cannot exceed 1. This is the case when the corpus has only one document and that document has no out-links that are part of the corpus.

Having described the mathematical formula for PageRank, it can be explained mathematically as,

The probability with which a ***random surfer*** *clicks on a random webpage*.

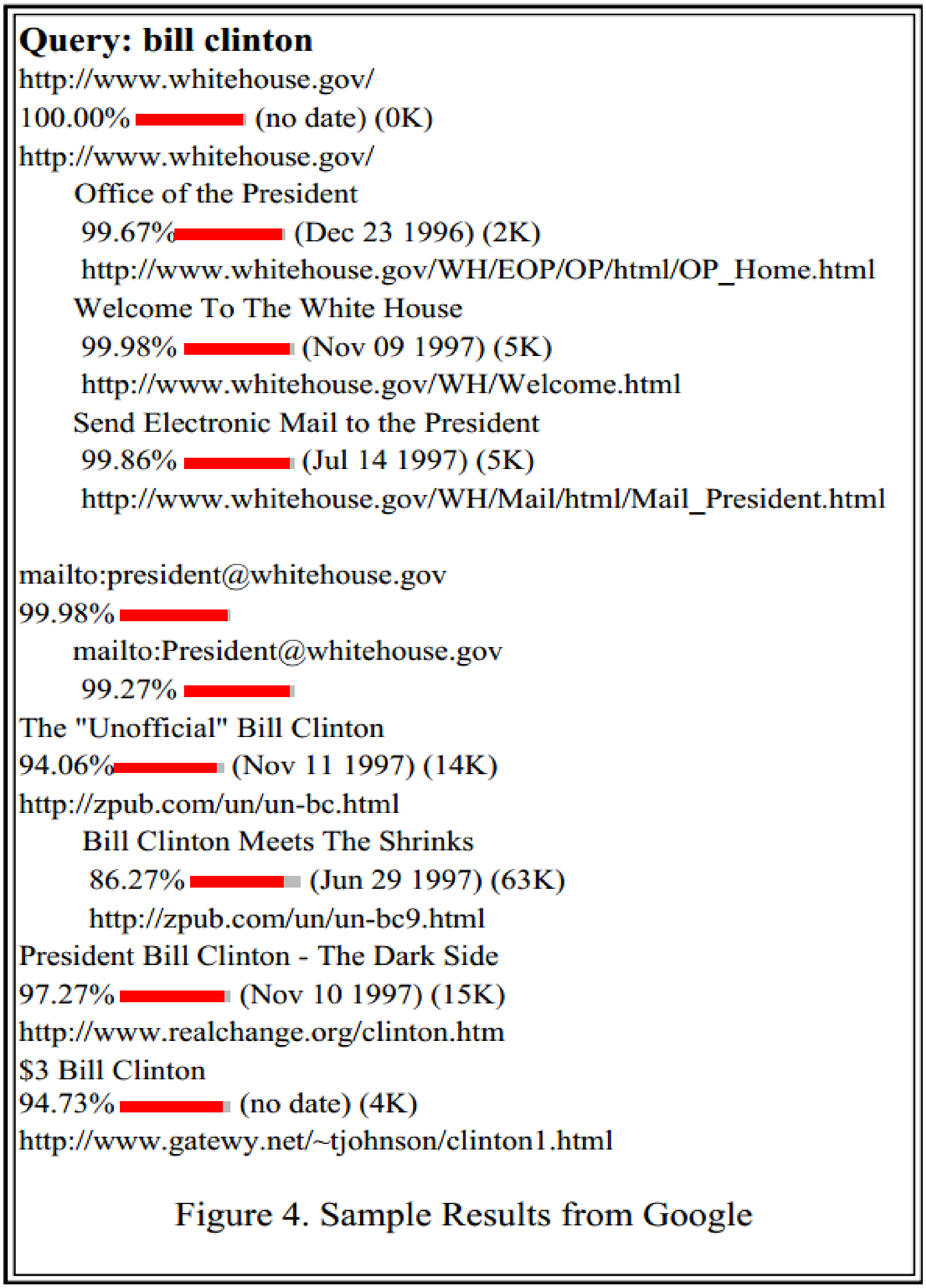
Intuitively, a page has higher page rank if it has a lot of in-links or some in-links with high page rank values. Similarly it has lower page rank if it has a lot of out-links.

The damping factor “d” was introduced to handle a common trait of human beings, Boredom. So “d” is the probability with which a **random surfer** gets bored of a webpage and jumps to another page.

To demonstrate the effectiveness of this algorithm, the authors have published following results for the query “bill clinton”[2]. The red color bars in the figure are the PageRanks of the respective links. We can see that the pages are displayed in decreasing order of their PageRanks. And as we see, the top result is from whitehouse.gov which is the official website of the president`s office. Since Bill Clinton was the then President of United States, people would obviously expect a link to that page for that query.

Also, in the top ten results, there are none that talk about Clinton without Bill or a Bill without Clinton. That shows how relevant the results are. Another noticeable feature of these results is that none of the links are broken. This is generally due to high page rank. Links that are broken are less likely to appear on the homepage of a very popular website like msn.com or Yahoo!.

Thus, this algorithm has proven very efficient for Google and it was observed that all other search engines that existed during that time produced results that were less accurate than these. PageRank and other striking features of this search engine have given it significant edge over other search engines.


Query: bill clinton

http://www.whitehouse.gov/
100.00% (no date) (0K)
http://www.whitehouse.gov/
    Office of the President
    99.67% (Dec 23 1996) (2K)
    http://www.whitehouse.gov/WH/EOP/OP/html/OP_Home.html
    Welcome To The White House
    99.98% (Nov 09 1997) (5K)
    http://www.whitehouse.gov/WH/Welcome.html
    Send Electronic Mail to the President
    99.86% (Jul 14 1997) (5K)
    http://www.whitehouse.gov/WH/Mail/html/Mail_President.html

mailto:president@whitehouse.gov
99.98%
    mailto:President@whitehouse.gov
    99.27%
The "Unofficial" Bill Clinton
94.06% (Nov 11 1997) (14K)
http://zpub.com/un/un-bc.html
    Bill Clinton Meets The Shrinks
    86.27% (Jun 29 1997) (63K)
    http://zpub.com/un/un-bc9.html
President Bill Clinton - The Dark Side
97.27% (Nov 10 1997) (15K)
http://www.realchange.org/clinton.htm
$3 Bill Clinton
94.73% (no date) (4K)
http://www.gatewy.net/~tjohnson/clinton1.html

Figure 4. Sample Results from Google

Figure 1.1 Sample Results of Google search engine[2]

Chapter 2 - Introduction to Hadoop

Apache Hadoop is a framework built to store and process arbitrarily large amounts of data on a distributed file system comprised of commodity hardware. It is an open source software built and used by developers across the globe. It is licensed under Apache License 2.0. It is written in Java programming language and command line utilities are written as shell scripts.

The term Hadoop was coined by Doug Cutting, who, along with Mike Cafarella created this technology. Hadoop came to life in February, 2006. It has its origins in Apache Nutch, an open source web search engine, which in turn is part of Apache Lucene project. Lucene is apparently the widely used search engine for text library.

Nutch project was started in 2002 and very soon they realized their architecture could not handle billions of pages on the web. Later, with influence from Google Distributed File System (GFS), they developed an open source implementation of their storage system called Nutch Distributed FileSystem (NDFS) in 2004. In the same year, Google introduced MapReduce to the world and the Nutch developers did not take a long time to migrate all their algorithms to run using MapReduce and NDFS.

Finally in 2006, these two technologies evolved to form a separate project called Hadoop, as a subproject of Lucene. Yahoo! Corporation provided a dedicated team to develop Hadoop into a full fledged web scaled system.

As quoted in the book "Hadoop- The Definitive Guide 2nd edition, in 2008, Hadoop created world record for sorting a terabyte of data in lowest possible time. It took 209 seconds to sort the data on a 910-node cluster. This time was further decreased significantly in the following years.

Hadoop is a term used to refer to a family of projects, all of which categorize as infrastructure for distributed computing and large-scale data processing. I am going to describe one of those projects MapReduce which is most relevant to my project.

MapReduce [3]

MapReduce is a parallel programming model to process data stored in HDFS (Hadoop Distributed File System). It can be implemented in several languages like Java, Ruby, C ++ etc.

Working of MapReduce

Each MapReduce job is comprised of a Map task and a Reduce task.

Mapper:

The job splits input into chunks of size ranging between 64 MB to 128 MB. Each split is processed by a map task. The number of map tasks depending on the number of input splits. The input and output of each task (map, reduce) is a Key-Value pair. The types of keys and values can be chosen by the programmer.

Each Mapper processes one block of input and it calls a map function for each line of input. Each line is sent as a key-value pair to the corresponding map function. The key is the line offset and the value is the text in the line. The key is ignored because it is not relevant to the data to be processed. Map processes the data and emits Key-Value pair(s) to the reducer. All the map tasks are performed in parallel which is the reason for such tremendously good performance.

Reducer:

As the name suggests, a reducer reduces a set of key-value pairs, in fact the set of values that have the same key. The outputs from one or several map tasks that emit key-value pairs with same key are all aggregated and sent to the same reducer as input. It processes the collection of values and emits the final answer.

Several reducers process input data in parallel and write to the same output directory specified in user arguments. The number of reducers is usually determined by number of unique keys that the map tasks emit but however a programmer can explicitly set the number of reducers. In the latter case, even the key-value pairs with same key can be split and go to several individual reducers. Increasing hardware usage increases efficiency in most cases. Hence setting number of reducers high yields better performance.

Each reducer emits output to a file called "part-r-0000*". * is the number of reducers -1. For example if there are 50 reducers, then the output files range is part-r-00000 to part-r-00049. Multiple output files for the same output does not cause any kind of problem when they are given as input to another map task as they are all considered as one whole input file present in the user-specified directory.

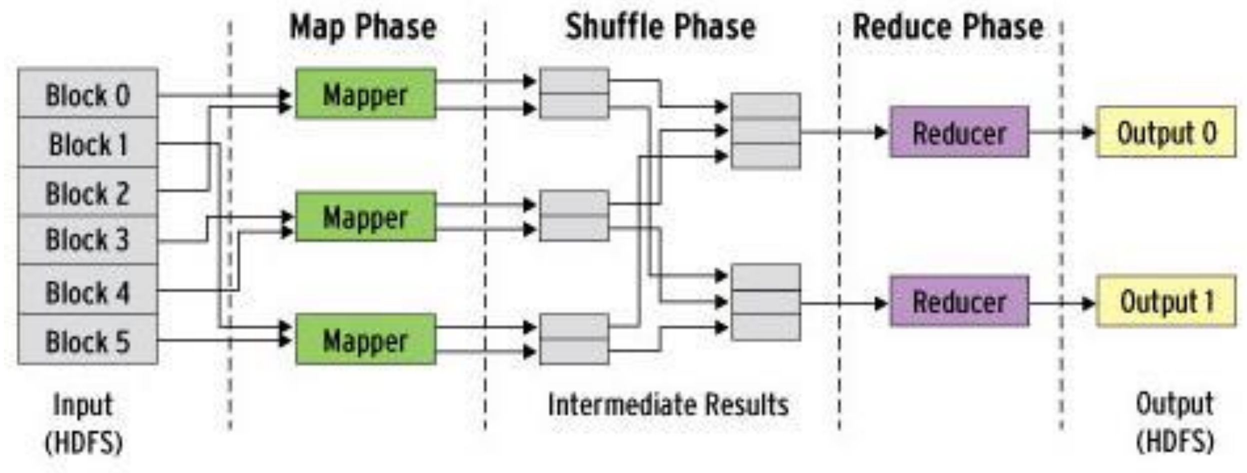


Figure 2.1 Flow of data through MapReduce Phase: http://www.admin-magazine.com/HPC/Articles/MapReduce-and-Hadoop

Implementation and Configuration Details of MapReduce

Mapper Implementation

Mapper<K1,V1,K2,V2> is an interface inside org.apache.hadoop.mapred package. Here K1,V1 represent the data types of input key-value pair and K2,V2 represent datatypes of output key-value pair. In java, which is the most common implementation language for MapReduce, it is implemented by a static Mapper class that calls a map function for each line of input. Each instance of a Mapper gets one input split as its input. Any global variables declared outside the map method can be accessed only from the map methods of this Mapper class. Since the whole purpose of MapReduce is parallel implementation, no global variables are declared to hold data that has to be accessed by all the mappers.

Signature:

```
public static class <Mapper Name> implements Mapper<K1,V1,K2,V2>
{

    public void map(K1 key,Iterator<V1> values,Context<K2,V2> context)
    throws Exception(s)

    {

         <Body>

    }

}
```

Depending on the version of the hadoop library, the signature can vary. For example the context object may be replaced by OutputCollector in a different Hadoop library.

Reducer Implementation

Reducer<K2, V2, K3, V3> is an interface in org.apache.hadoop.mapred package. K2,V2 are used to represent that they are the output key-value pair of the mapper. K3,V3 are the data types of the key and value of output of reducer. A static reducer class implements this Reducer interface.

Signature:

```
public static class <Reducer Name> implements Reducer<K2,V2,K3,V3>
 {

     public void reduce(K2 key,Iterator<V2> values,Context<K3,V3> context)

    throws Exception(s)
    {
         <Body>
    }

 }
```

Configuration of MapReduce job

Configuring the map and reduce tasks is done in the main method that invokes these tasks. In order for that, a Configuration object and a JobConf/Job object are constructed. These classes have methods that can be called to configure the map and reduce tasks.

The number of mappers is by default equal to the number of input splits. However the JobConf class has a method to set the number of mappers explicitly. This can only be given as a hint to the framework. It need not implement that number of mappers. If a user is explicitly setting number of mappers, he should remember to set that number atleast as high as the number of input splits.

The number of reducers is by default equal to one of the following:

[0.95 * (nodes * mapred.tasktracker.tasks.maximum)] or

[1.75 * (nodes * mapred.tasktracker.tasks.maximum)]

In the first case, all the reducers can launch immediately and process map outputs as the maps finish. Latter case handles load balancing very efficiently. Here the faster nodes finish their first round of reduces and launch a second round of reduces.

While those are the default values for number of reducers, a programmer can overwrite this number by setting a number for himself. Generally, higher the number of reduces better is the performance. So it makes no sense to set it to a number less than the default unless there is a purpose.

Besides the number of mappers and reducers, we can also configure the input/output key class, input/output value class, and many more parameters that best suit the application. Job specific parameters are set using Job/JobConf object while the environment parameters like the heap space of jvm(Java Virtual Machine),buffer size for the input and output can be set using configuration object.

Hadoop Distributed File System [4]

Hadoop Distributed File System (HDFS) is one of the four modules of Hadoop. It is a distributed file system which provides high aggregate bandwidth across the cluster of machines. It is usually used to store large amounts of data. The data is distributed across several machines connected in a cluster and the hadoop commands are used to access this data. These commands are used to transfer data between HDFS and local file system, read the data using runnable hadoop jars that contain map reduce jobs, etc. Data stored inside HDFS cannot be accessed via local file system without using hadoop command line tools.

NameNode and DataNodes

HDFS has a master/slave architecture comprised of a single NameNode and several DataNodes. The NameNode manages the file system namespace and regulates access to files by clients. It is central for all the nodes in the cluster. On the other hand, each node in the cluster has a unique DataNode which manages the data distributed to this node. Each can hold a minimum data chunk of size 64 MB. The NameNode manages the mapping of these data chunks to the DataNodes, i.e., which DataNode holds which particular data chunk is known to NameNode. executes file system namespace operations like opening, closing, and renaming files and directories. The DataNodes are responsible for serving read and write requests from the file system's clients are served by DataNodes and all the namespace operations are implemented by NameNode.

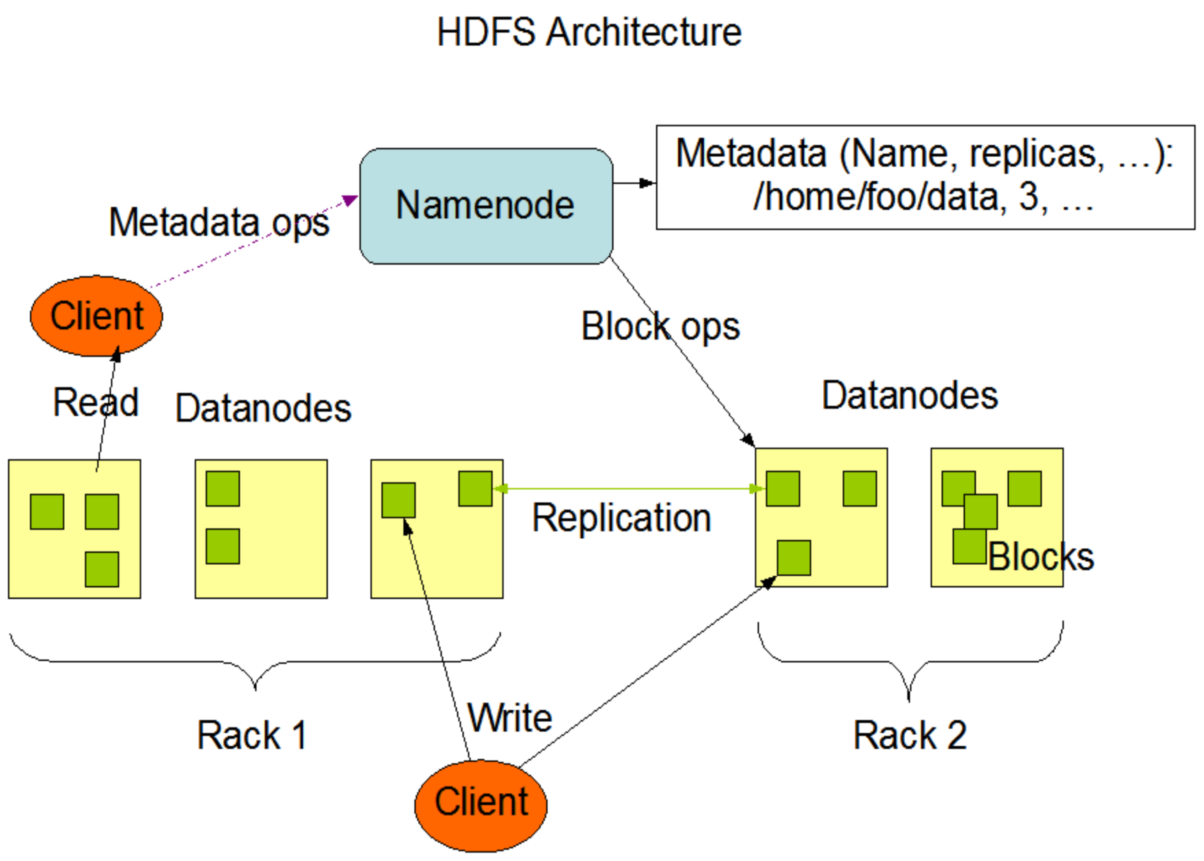


Figure 2.2: Illustration of HDFS master/slave architecture: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Maintaining the NameNode is crucial to a HDFS system because it has the directory tree of all files in the system. Any failure of NameNode can results in failure of HDFS. In order to prevent such a situation, there is an option to maintain a secondary NameNode on a different machine so that if the primary fails, secondary can keep the cluster running.

Chapter 3 - Implementation Details

The implementation details of the project can be broken down into several steps.

- Initial Idea of the project
- Data Acquisition and Transformation
- Creating Wikipedia database in MySQL
- Generating Input data for PageRank algorithm
- Build the algorithm
- Testing the algorithm

Initial Idea of the project

Initially, the idea for the project was to analyze historical Twitter feed of a period of three months when some natural disaster has taken place. For this purpose, I explored Twitter REST API to fetch tweets of a period of three months from the past. However I had realized that Twitter API does not provide data older than seven days from the current date.

However none of the APIs available to that day provided free data and their costs were way too expensive. Then I was advised to use Wikipedia data and build a PageRank algorithm to run on that data.

Data Acquisition and Transformation

MediaWiki API provides all the Wikipedia data at no cost. I fetched all the English Wikipedia data that was close to 36 GB and used Java-based Wikipedia Library (JWPL) to run transformation of this data set. This library provides jar files which I ran from the terminal to transform the data files downloaded into eleven text files that are in relational database table format. The transformation process took several hours due to the large size of the data set. The output files generated were all in table structure which could be directly dumped into a database without any more cleaning.

Wikipedia Database

The files generated in the above step were then uploaded into MySQL server of the CIS department. Following are the tables and their columns.

Category_outlinks

```
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| id       | bigint(20) | NO   | MUL | NULL    |       |
| outLinks | int(11)    | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

MetaData

```
+------------------------+-------------+------+-----+---------+------

| Field                  | Type        | Null | Key | Default | Extra          |

+------------------------+-------------+------+-----+---------+------

| id                     | bigint(20)  | NO   | PRI | NULL    | auto_increment |

| language               | varchar(255) | YES  |     | NULL    |                |

| disambiguationCategory  | varchar(255) | YES  |     | NULL    |                |

| mainCategory           | varchar(255) | YES  |     | NULL    |                |

| nrofPages              | bigint(20)  | YES  |     | NULL    |                |

| nrofRedirects          | bigint(20)  | YES  |     | NULL    |                |

| nrofDisambiguationPages | bigint(20)  | YES  |     | NULL    |                |

| nrofCategories         | bigint(20)  | YES  |     | NULL    |                |

| version                | varchar(255) | YES  |     | NULL    |                |

+------------------------+-------------+------+-----+---------+----------------+
```

Page

```
+------------------+--------------+------+-----+---------+----------------
| Field            | Type         | Null | Key | Default | Extra          |
+------------------+--------------+------+-----+---------+----------------+
| id               | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| pageId           | int(11)      | YES  | UNI | NULL    |                |
| name             | varchar(255) | YES  | MUL | NULL    |                |
| text             | longtext     | YES  |     | NULL    |                |
| isDisambiguation | bit(1)       | YES  |     | NULL    |                |
```

PageMapLine

```
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name   | varchar(255) | YES  | MUL | NULL    |                |
| pageID | int(11)      | YES  |     | NULL    |                |
| stem   | varchar(255) | YES  |     | NULL    |                |
| lemma  | varchar(255) | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
```

Category_inlinks

```
+---------+------------+------+-----+---------+-------+

| Field   | Type       | Null | Key | Default | Extra |

+---------+------------+------+-----+---------+-------+

| id      | bigint(20) | NO   | MUL | NULL    |       |

| inLinks | int(11)    | YES  |     | NULL    |       |
```

Category

```
+--------+--------------+------+-----+---------+----------------+

| Field  | Type         | Null | Key | Default | Extra          |

+--------+--------------+------+-----+---------+----------------+

| id     | bigint(20)   | NO   | PRI | NULL    | auto_increment |

| pageId | int(11)      | YES  | UNI | NULL    |                |

| name   | varchar(255) | YES  | MUL | NULL    |                |

+--------+--------------+------+-----+---------+----------------+
```

Category_pages

```
+-------+------------+------+-----+---------+-------+

| Field | Type       | Null | Key | Default | Extra |

+-------+------------+------+-----+---------+-------+

| id    | bigint(20) | NO   | MUL | NULL    |       |

| pages | int(11)    | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

Page_categories

```
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| id    | bigint(20) | NO   | MUL | NULL    |       |
| pages | int(11)    | YES  |     | NULL    |       |
```

Page_inlinks

```
+---------+------------+------+-----+---------+-------+
| Field   | Type       | Null | Key | Default | Extra |
+---------+------------+------+-----+---------+-------+
| id      | bigint(20) | NO   | MUL | NULL    |       |
| inLinks | int(11)    | YES  |     | NULL    |       |
```

Page_outlinks

```
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| id       | bigint(20) | NO   | MUL | NULL    |       |
| outLinks | int(11)    | YES  | MUL | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

Page_redirects

```
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| id        | bigint(20)   | NO   | MUL | NULL    |       |
| redirects | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

The diagrammatic representation for the schema

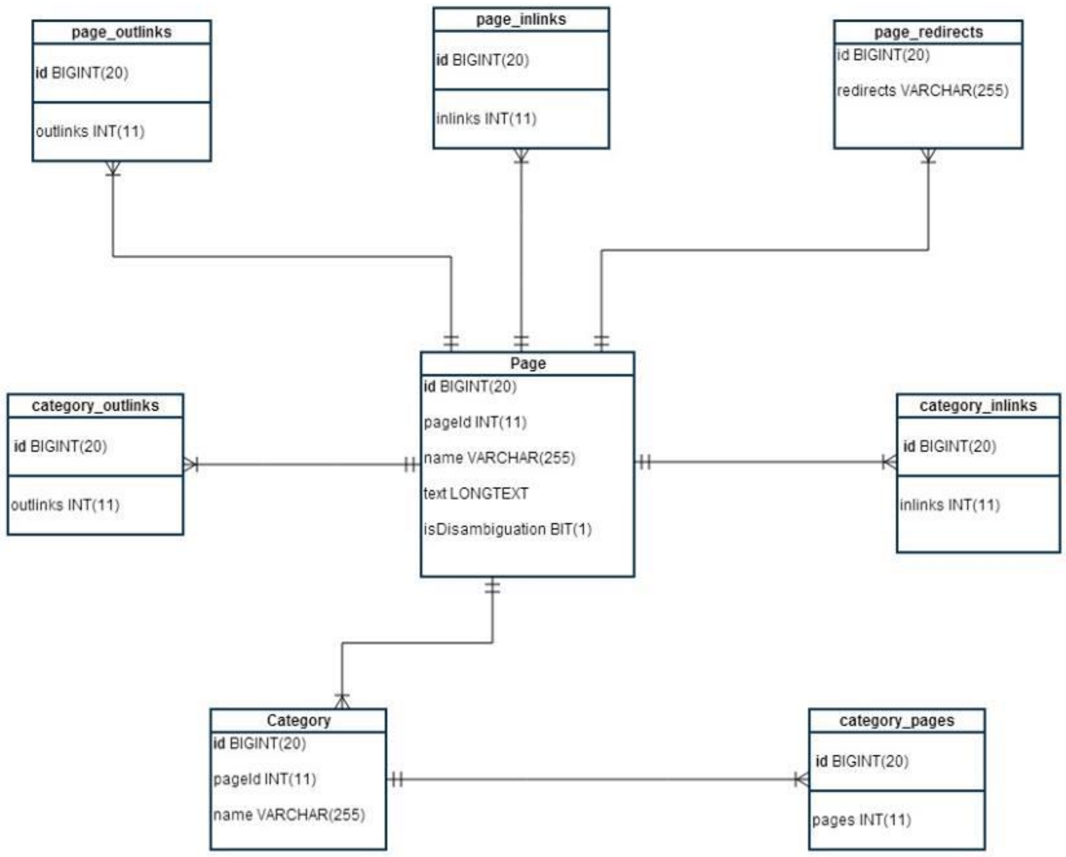


Figure 3.1.1 Database Schema for the implementation

Generating input for PageRank algorithm

All the above tables have foreign key relations and querying into that database would give me the pages with their out links and in links. But even after indexing the tables, each query was taking forever to display results and so I adopted the MapReduce approach to create input for the PageRank algorithm.

One of the files obtained after transformation contained title of a page and its content. That was the first input file for my map reduce jobs.

First MapReduce job

This job read the above file and added xml tags <page>, <text> and <title> to each line of the file. Here each line was a title and its contents.

For example, this job`s output has the following format.

<page><title> Title <\title> <text>Text<\text><\page>

Second MapReduce job

The input to this job is the output of the previous job. It will parse each line of the input file and retrieve title and text of each line that are between the tags title and text.

The text in each line is processed for outlinks for that page. Here outlink refers to only those links that redirect to other Wikipedia pages. Other links are ignored. Each outlink is enclosed in square parenthesis ([[ ]]). There are several kinds of outlinks that are contained in these parenthesis. Some are themselves titles in Wikipedia, some are redirect pages and some are nested links. For the current project, I have only considered actual titles since they become part of the link graph.

The reducer identifies all the links that are not titles in Wikipedia and ignores them. So this job emits title of a page and all its outlinks that are in turn titles of other Wikipedia pages.

Third MapReduce job

This job`s input is the output of previous job and output is an adjacency list. The output key of this job is the page title and the value is a list of outlinks of that title.

Building the Algorithm

PageRank algorithm is built in two stages. The first stage simply creates input to the second job in the following format:

```
A      [PR(A),{outlink1,outlink2,outlink3…outlinkn}]

B      [PR(B),{outlink1,outlink2,outlink3…outlinkm}] and so on.
```

Here A refers to a title and PR(A) refers to the initial page rank of page A. It is said to 1 for all pages. So for a page A, containing links x,y,z the reducere emits the following key,value pair:

```
A      [1,{x,y,z}]
```

A is the key and the other string is the value. They are tab separated (which is the default separator for key-value in Hadoop framework).

The second step contains four MapReduce jobs. These jobs iteratively run till all the page ranks converge. Each job takes output from the previous one in a cyclic order. So the first job`s input for the first iteration is from the first step described above, but for all the other iterations, it the output of the fourth map-reduce job that constitutes this step.

The following description explains what each of the four jobs mentioned above do.

First job distributes the page rank of that particular page to all its outlinks equally. Its mapper emits each two types of outputs. One is each outlink and its rank. This rank is therefore the original rank of the page (the key string) divided by total number of outlinks. The second output is and outlink and the title.

Its reducer makes a list of all the ranks and all the in-links of a given key. (Here the key is an outlink and value is either a rank of its in-link since the mapper emits a page and outlink in reverse order).

The second job reads the above output and also output from a separate path that contains the total corpus size.Corpus size is the total number of unique nodes in this link graph which is calculated in a separate MapReduce job before these iterations. In other words it’s the total number of titles in the Wikipedia data set. Then it applies the PageRank formula given by Larry Page and Sergey Brin.

$$\frac{\varepsilon}{N} + (1-\varepsilon)\sum_{i=1}^{m} S_m$$

Figure 3.2.2 Formula to calculate PageRank: Information Retrieval material by Dr. Doina Caragea.

In the above formula, N refers to the total corpus size and $S_{1, \ldots} S_m$ refer to the incoming page ranks of the given page. m is the total number of inlinks for the page. The constant factor ε is called the Damping Factor. It is set to the value 0.15.

Third and fourth jobs normalize the newly calculated page ranks of all pages by dividing them with their total sum. This resultant value is the new page rank for the second iteration. The iterations continue until these page ranks converge.

In each iteration, the number of input and output records of all the map and reduce tasks of all jobs should be the same as in the previous iterations. The figure below summarizes the input/output of iterations.

$$\text{Map input: } \left(Y, [PR(Y), \{Z_1, \ldots, Z_n\}]\right)$$

$$\text{Map output: } \left(Z_i, \frac{PR(Y)}{n}\right), \left(Y, \{Z_1, \ldots, Z_n\}\right)$$

$$\text{Reduce input: } \left(Y, [S_1, \ldots, S_m, \{Z_1, \ldots, Z_n\}]\right)$$

$$\text{Reduce output: } \left(Y, \left[\frac{\varepsilon}{N} + (1-\varepsilon)\sum_{i=1}^{m} S_m, \{Z_1, \ldots, Z_n\}\right]\right)$$

Figure 3.2.3 Formula to calculate PageRank: Information Retrieval material by Dr. Doina Caragea

Chapter 4 - Sample Results

In the first step, each line is raw wikipedia text is read and added xml tags.

Step1 Input Format:

```
12	12	Anarchism	{{Redirect|Anarchist|the fictional character|Anarchist
(comics)}}\n{{Redirect|Anarchists}}\n{{pp-move-indef}}\n{{Anarchism sidebar}}\n\n\'\'\'Anarchism\'\'\' is a
[[political philosophy]] that advocates [[stateless society|stateless societies]] based on non-
[[Hierarchy|hierarchical]] [[Free association……..such that they find objectsionable, but ….}}

25	25	Autism	{{dablink|This article is about the classic autistic disorder; some writers use the word
\'\'autism\'\………………… [[pervasive developmental disorder]]………………..}}
```

The output format of this step is

```
<page> <title>A</title><text>{{Two other uses|the letter of the alphabet|the English indefinite article|a and
an}}\n{{Use dmy dates|date=June 2013}}\n{{Technical reasons|A#|A-sharp|A??? (disambiguat…………
</text></page>

<page> <title>ASCII</title><text>{{distinguish2 | MS [[Windows-1252]], also known as \"ANSI\", or other types of
[[Extended ASCII]], often just called \"ASCII\"}}\n{{about|the character encodin}}………………….. </text></page>
```

The **second and third steps** together build the adjacency list. The input to the second is the output of the first described above. The output is as follows.

```
300		[Venice, 257, Christianity, China, Anderitum, Armenia, 300, Buddhism, Jews, Asanga, 278, Palmyra, India,
India, Mahayana, Lullingstone, Christian, 232, Mexico, Belize, Rome, Croatia, Franks, 330, Constantinople, Mesoamerica,
Panchatantra, 318, 1204, 221, Guatemala, Sanskrit, Yogacara, 343, Tikal]

30th_century_BC	[Ireland, Yuyao, Hangzhou, Caral, Djet, Djet, Ecuador, Neolithic, Anedjib, Stonehenge, Mesopotamia,
Mesoamerica, Troy, Americas, Americas, Shekel, Khafajah, 30th_century_BC, Pharaoh, Merneith, Zhejiang, Japan, Baghdad,
China, Djer, Djer, Djer, Iraq, Egypt, Egypt, Egypt, Egsypt, Semerkhet, Semerkhet]
```

The **fourth step** creates initial page rank for the final steps. Its input is the above output and its output is as follows:

```
300		[1,{Venice, 257, Christianity, China, Anderitum, Armenia, 300, Buddhism, Jews, Asanga, 278, Palmyra, India, India, Mahayana, Lullingstone, Christian, 232, Mexico, Belize, Rome, Croatia, Franks, 330, Constantinople, Mesoamerica, Panchatantra, 318, 1204, 221, Guatemala, Sanskrit, Yogacara, 343, Tikal}]

30th_century_BC	[1,{Ireland, Yuyao, Hangzhou, Caral, Djet, Djet, Ecuador, Neolithic, Anedjib, Stonehenge, Mesopotamia, Mesoamerica, Troy, Americas, Americas, Shekel, Khafajah, 30th_century_BC, Pharaoh, Merneith, Zhejiang, Japan, Baghdad, China, Djer, Djer, Djer, Iraq, Egypt, Egypt, Egypt, Egypt, Semerkhet, Semerkhet}]
```

The final steps iteratively calculate page rank until the page ranks converge. Its output is similar to the above but the page ranks values are different.

The final step is to sort the titles by their page rank values and display top ten results in descending order. Input to this step is the output of the above class and its output is the final result of the algorithm.

```
India	4813294.0
France	4787018.5
Germany	4560347.5
Italy	3644442.0
Latin	3599026.0
Netherlands	2916263.5
Europe	2887972.0
Russia	2864662.2
Islam	2785769.0
China	2643788.2
```

The ranks we see above are large real numbers because the page rank values have been multiplied by a constant factor so that it makes the difference between ranks quite clear.

Chapter 5 - Performance Analysis

Before explaining the performance analysis in my project, it is good to throw some light on the differences between Scaling-Up and Scaling-Out.

Scaling Up

In simple terms, scaling up is nothing but increasing the hardware and computing power of a single machine so that it can handle large-scale data processing. Upgrading the processor of a machine, increasing its RAM(Random Access Memory) etc are examples of scaling up. Although this is effective to some extent, any machine has a threshold unto which it can be scaled up. Beyond that, you cannot increase the computing power of that machine. With massive explosion of data this no longer is a good option. Although, it is possible to build super computers with extremely high computing capability, if it crashes for some reason, the entire data processed/unprocessed is lost.

Scaling Out

Scaling out is utilizing a network of computers(connected) for large-scale data processing. In this, many machines that are not necessarily high-end are connected in a network. The data is replicated/distributed across all these machines and the processing is split so that each machine will only handle part of the entire load.

MapReduce is one such distributed computing model which has more pros than cons as compared to a parallel relational DBMS. It seemed to be the best approach for processing the Wikipedia data set in my project since the data set was reasonably big, given the available resources.

Performance comparison between MySQL and Hadoop MapReduce observed:

- Loading the entire data set into MySQL server on the CIS Linux took several hours to reach completion. On the other hand, it took less than an hour to transfer all that data into HDFS.
- Indexing the tables in MySQL was the only way to query such large tables in reasonable time. But indexing itself took forever.
- MapReduce framework splits the input file into several fixed size splits each of 64 MB by default. It is also possible to set the split size different unto a maximum of 128 MB. It then scales out the processing to several nodes by allocating each input split to a unique Mapper object. This way the Map phase is executed in parallel by several Mappers on input splits that are reasonably small. The first job I run was to add xml tags to the input file. The input to this job was approximately 26 GB and the map phase took only a few minutes to finish. This shows how efficient this is when compared to querying in MySQL.
- The reduce phase also is executed in parallel by a number of reducers and this phase takes approximately the same amount of time as the map phase. The time taken by each phase however depends on the processing logic inside the Mapper and Reducer.
- For even better performance, the number of mappers and reducers can be manually set to values higher than their defaults. I set the final job`s number of reducers to 1 because I wanted the output to be saved to only one file in HDFS. In this case, if the input is small enough for one single reducer to process, then there will not be any performance issues but otherwise it is also suggested to allocate as many reducers as possible.
- The total number of Mappers and Reducers available on the Beocat cluster are 160 each. So that is the maximum number of parallel tasks that can be run in this cluster. If more are

allocated by the programmer, for example 200 reducers, the reduce tasks are queued and executed one after the other in some of the reducers. However, logically it looks like all 200 are executed by separate reduce tasks since there will be 200 output files created in the output folder.

The performance difference observed when the number of reducers used is 1 versus when the number of reducers used is 100 is very significantly noticed as seen below.

Number of Reducers Used	Time taken by Reduce Phase
1	1440 seconds
25	495 seconds
50	340 seconds
100	120 seconds

Table 5.1 Change of Time taken with change in number of reducers

The above table was created by running a map reduce job to add xml tags to the input page. This job`s input size was 26 GB. All other jobs took inputs that were relatively lesser in size compared to this one and so I tested performance on this job.

As it is seen from the table, the difference in the time taken by the reduce phase of the job is very significant when 1 reducer is used and when 100 reducers is used. The times differ by a factor of 11. This demonstrates why a distributed multi-node computing system is more efficient in terms of time as compared to a single-node system.

Chapter 6 - Testing

Software Testing is as important a phase as development phase or any other phase in the software development life cycle. However good programmers a software development team has, they are ultimately human and so mistakes seep in. Depending on the length and breadth of a software application, scope of failures increase. Software Testing can help detect these failures and fix them before a product goes into production.

Software Testing has several classifications among which Unit Testing is the most basic type.

Unit Testing

Testing should be done in the early stages of SDLC as the cost of detecting a bug is cheaper when it is done in the early stages. Unit Testing is the type of software testing that is done on each individual module of a software. It is the most common testing technique that most software companies perform.

Several unit testing frameworks have been developed for all programming languages. Some examples are JUnit, JWalk for Java, ASPUnit for ASP etc. The recently famous "Agile" methodology has its foundation in unit testing.

For this project, I describe Unit testing as testing each class that is part of the algorithm for its expected output. There are totally seven classes. Each class has its own input and output. And the algorithm is built such that these classes are all executed as a chain. The output of the first class is input to second and so on. So even if one class does not give the desired output, it breaks the whole algorithm.

For example, the first class has MapReduce job that parses the original input file and adds tags to it. If some of the lines do not get tagged properly, those will be ignored by the next job and so those titles were never be considered for calculation of PageRank. That will ultimately give me wrong results.

Another most crucial test is on the values (ranks and outlinks) for a given key(title). There are pages that are named by numbers like “1947”. If the program understands this title as a rank rather the page name, it adds that value to the total rank of that page. That will result in wrong calculation of PageRank for that page.

Each class has atleast one MapReduce job. So unit testing involves testing each MapReduce job for its desired output. These outputs are all written to separate folders in hdfs. To verify the correctness of output, first the folders have to be copied to local file system and only then can we open the output file to read the output. Files present in the folders inside HDFS cannot be read. They are only Write-Only.

Another important test of the MapReduce jobs is the running time of each. The whole purpose of using Hadoop framework is achieve time efficiency by way of parallel computing. So even the execution time for each mapreduce job is observed. If a particular job takes longer time than it should, one way to fix that is allocate more mappers/reducers to it.

Integration Testing

The purpose of Integration Testing is to see how individual modules that passed unit tests correctly perform together. So integration testing is defined as testing the functionality of two or more modules of a software together.

Most effective way to build a software application is to split functionality into several modules and build each module separately so that they do not significantly break each other`s functionality even if they are dependent modules.

In such a scenario, once the modules are unit tested, these are integrated progressively. At each step of integration, the Integration Testing is performed to check the desired output of the combined system. If at any given point results deviate from their desired state, it would be easy to identify which part of the system is breaking it or producing wrong results.

In context of the current project, as I mentioned earlier, each MapReduce job`s output is input to atleast one other job. So when a driver class calls these two jobs consecutively, they should function exactly as though they were separately executed. If not, something is wrong in the way my program is reading input from the previous job or in the configuration.

A number of MapReduce jobs run in a sequence for several iterations for calculation of pagerank and normalizing these values. The number of input records and output records should be the same at the beginning and end of each iteration respectively. If this fails, something is wrong in one of my jobs. So this can be tested with Integration Testing. And these iterations are apparently the most important part of the algorithm.

System Testing

Another testing strategy that is critical to the current project is System Testing. System Testing is a kind of Black-box testing where the system as a whole is tested for its requirements and desired output. Internal design of the system is not considered in this strategy.

A driver program that invokes each and every job the specified number of times in the required sequence is written and this program is passed the input file. A number of arguments are given as part of the command to run this program. These arguments are the paths to intermediate output folders. Final output folder should contain titles with top ten page ranks in descending order.

If this test is performed directly without Unit Testing and Integration Testing, there is very high probability of the system failing to meet its requirements. And it becomes ridiculously expensive to detect bugs if the system is decently large. Therefore all three testing strategies described above are equally important.

Regression Testing

Regression Testing is done to identify bugs whenever an already built software application undergoes some changes like in configuration or an additional patch.

When analyzing the performance of each MapReduce job on the data that I passed as input to it, there were several performance issues like JVM running out of heap space or reducer taking forever to start up. These issues are usually due to not configuring the job correctly.

So whenever such an issue is discovered, I recoded the jobs by making changes like using different data structures, configuring more number of reducers, outputting different key-value pairs, partitioning the intermediate key-value pairs differently than the default.

When such a change is made, program tends to perform differently. Although it is expected to perform better, there might be a bug that causes it to deviate from the expected behavior. Regression Testing helps identify these bugs.

As an example scenario in my project, any algorithm`s performance depends on the size and format of input data. Although the algorithm ran very well on the input data the first time, I optimized some of the jobs to include some noisy data as well to see if the precision of the results could improve further, and this led to Java Heap space errors and also exceptions like StringOutOfBounds, ArrayIndexOutOfBounds etc. These were due to the difference in the format of the input lines that I read this time from the lines that I read before. These kind of bugs, if untested will result in either breaking the algorithm completely or wrong results.

Chapter 7 - Conclusion and Future Scope

This project ranks all Wikipedia pages to display most relevant results to the user based on the query. The ranking is based on the number of in-links and out-links of a page. Google uses this algorithm (and many others) to rank web pages. It can be extended to implement on various other domains. For example, Recommender systems are emerging technology in the e-commerce world and they are giving unique edge to the companies like Amazon and YouTube. This idea of this algorithm can be used and a new algorithm can be built that ranks items in various domains.

Google currently ranks documents in the web and displays the links as results to its users. In future, it could aggregate information in these links that is relevant to the user query and present that information directly instead of links. Although this idea many times makes a user`s life easy, it proves detrimental to the search engines and the resources like StackOverflow that are among the top ranked websites by Google and many other search engines. So it is not a commercially viable idea. There could, however be some unique applications that could use this idea.

References

[1] Lawrence Page, Sergey Brin, Rajeev Motwani and Terry Winograd: The PageRank Citation Ranking: Bringing Order to the Web. *Stanford Infolab*,1999.

[2] Sergey Brin and Larry Page. The anotomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*. 30 107-117, 1998.

[3] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung and Bongki Moon: Parallel Data Processing with MapReduce: A Survey. *SIGMOD Record*, 2011.

[4] Flow of data through MapReduce Phase:

http://www.adminmagazine.com/HPC/Articles/MapReduce-and-Hadoop

[5] HDFS Architecture: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[6] ER Diagram: http://upload.wikimedia.org/wikipedia/commons/thumb/4/42/MediaWiki_1.20_%2844edaa2%29_database_schema.svg/2500px-MediaWiki_1.20_%2844edaa2%29_database_schema.svg.png

[7] Unit testing: http://en.wikipedia.org/wiki/Unit_testing

[8] System Testing: http://www.softwaretestingclass.com/system-testing-what-why-how/