A

PROJECT REPORT

ON

Information Retrieval from Wikipedia Dataset

B.Tech (CE) Sem-VI

In fulfilment of all requirements for
Bachelor of Technology
In
Computer Engineering
Submitted by

Sagar P Patel (CE- 095)
Vimox S Shah (CE- 109)

Under the Guidance of

Prof. Prashant M. Jadav

DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF TECHNOLOGY,
DHARMSINH DESAI UNIVERSITY
COLLEGE ROAD, NADIAD- 387001
April 2016

DHARMSINH DESAI UNIVERSITY

NADIAD-387001, GUJARAT

CERTIFICATE

This is to certify that the project carried out in the subject of System Design Practices entitled "***Information Retrieval from Wikipedia Dataset***" and recorded in this report is a bonafide work of

1) Sagar P Patel	ROLL NO: CE-095	ID: 13CEUOG086
2) Vimox S Shah	ROLL NO: CE-109	ID: 13CEUOF105

Of Department of Computer Engineering, semester VI. They were involved in Project developing during the academic year 2015 -2016.

Prof. Prashant M. Jadav	**Dr. C.K.Bhensdadia**
Project Guide,	Head,
Department of Computer Engineering	Department of Computer Engineerinng
Faculty of Technology	Faculty of Technology,
Dharmsinh Desai University, Nadiad	Dharmsinh Desai University, Nadiad

Table of Contents

Abstract....4
Introduction....5
Software Requirement Specifications....9
Design....11
Implementation Details....16
Testing....33
Screen Shot of Information Retrieval System....37
Conclusion....39
Limitation and Future Extension:....40
Bibliography....41

Abstract

The aim of this project is about Querying and Retrieving documents from Wikipedia dump. In technical term it is called information Retrieval(IR) System. The standard approach to information retrieval system evaluation revolves around the notion of relevant and non-relevant documents. Implementation of various algorithm for checking most relevant document from large amount of data gives effective information retrieval system. So using information retrieval systems we define specific queries and then answer these specific queries.

Chapter 1

Introduction

Information retrieval did not begin with the Web. In response to various challenges of providing information access, the field of information retrieval evolved to give principled approaches to searching various forms of content. The field began with scientific publications and library records, but soon spread to other forms of content, particularly those of information professionals, such as journalists, lawyers, and doctors. Much of the scientific research on information retrieval has occurred in these contexts, and much of the continued practice of information retrieval deals with providing access to unstructured information in various corporate and governmental domains.

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) There are many Information retrieval systems. Information retrieval has developed as a highly empirical discipline, requiring careful and thorough evaluation to demonstrate the superior performance of different novel techniques used by IR systems on representative document collections.

The group of documents over which information retrieval is performed is called as collection. It is also called as Document corpus. User submits query. Query is what the user conveys to the computer in an attempt to communicate the information need. Information need is the topic about which the user desires to know more.

Documents retrieved by IR systems are arranged according to their relevance to a given search query. A document is relevant if it is one that the user perceives as containing information of value
with respect to their personal information need. There exist many IR systems like Google, Bing, MSN and many more.

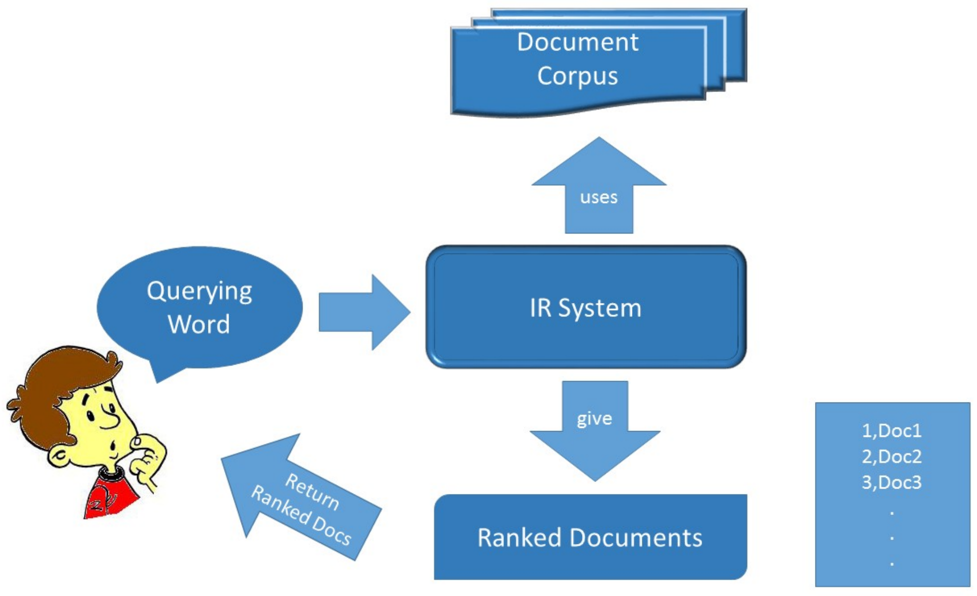


Fig. 1.1 Typical Information Retrieval Task

We implemented information Retrieval System for the Wikipedia dump.For that we used Big Data Technologies.We used Hadoop Framework for processing and storing large amount of data.We used Hbase as our data store.As Frontend we used JSP(Java Server Pages) and Javascript for GUI.

Brief introduction to Big Data and Related Technologies

★ What is Big Data?

Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn’t fit the structures of your database architectures. To gain value from this data, you must choose an alternative way to process it. The hot IT buzzword of 2012, big data has become viable as cost-effective approaches have emerged to tame the volume, velocity and variability of massive data. Within this data lie valuable patterns and information, previously hidden because of the amount of work required to extract them. To leading corporations, such as Walmart or Google, this power has been in reach for some time, but at fantastic cost. Today's commodity hardware, cloud architectures and open source software bring big data processing into the reach of the less well-resourced.

Characteristics of Big Data 4V's:

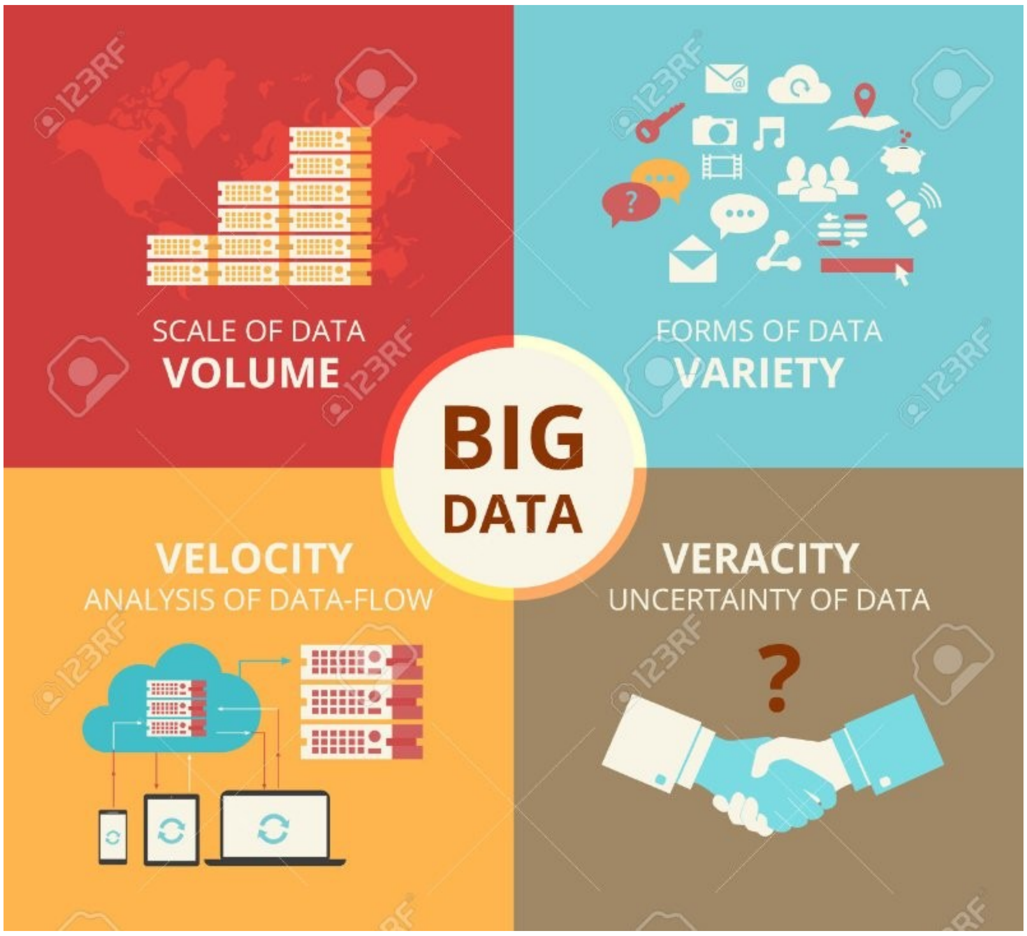


fig 1.2 characteristics of Big Data

★ What is Apache Hadoop?

Hadoop is the leading open-source software framework developed for scalable, reliable and distributed computing. With the world producing data in the zettabyte range there is a growing need for cheap, scalable, reliable and fast computing to process and make sense of all of this data. Hadoop is built just for that as it runs on a large number of machines that share the workload to optimise performance. Moreover, Hadoop replicates the data throughout the machines ensuring that the processing of data will not be disrupted if one or multiple machines stop working.

★ What is Apache Hbase?

Hbase is a non-relational columnar database that runs on top of HDFS(Hadoop Distributed File System). This schema-less database supports in-memory caching via block cache and bloom filters that provide near real-time access to large datasets, making it especially useful for sparse data which are common in many Big Data use cases. However, it is not a replacement for a relational database as it does not speak SQL, support cross record transactions or joins.

- ✔ Technologies/Framework Used: Java EE(Servlet & JSP), Hadoop, Hbase(as Data Store), Oozie(as Workflow Engine)
- ✔ Data Visualization Tools Used: Tableau (Business Intelligence Tool for Visualization) ,
  Gephi(Open Source tool for Network Analysis and Visualization)
- ✔ Diagram Tool Used: Start UML

Chapter 2

Software Requirement Specifications

6.1) Search Engine like GUI

Input: User type web url in addressbar of browser

Output: User can see nice GUI

6.2) Single word query result

Input: User can type single word query

Output: User can see most relevant links of wikipedia pages to his/her query

Precondition:

User visits the web application and is on the home page.

User has once searched for something on the application and is on the result page.

Post Condition:

Web application provides user with result page and waits for the next query given to the system.

Exception:

I. Query term could not be found in the corpus.
II. Null or Special Character (Except query with prefix ‘**’) Query term search.

6.3) I'm Felling Lucky” type Functionality

Input: User can type any word and click on “Test Your Luck” button

Output: User redirected first result of search page

Precondition:

User visits the web application and is on the home page.

User has once searched for something on the application and is on the result page.

Post Condition:

Web application redirects user to the Wikipedia Title page with highest relevance from the result set.

Exception:

I. Query term could not be found in the corpus
II. Null or Special Character (Except query with prefix '**') Query term search.

6.4) Special search feature

Input: User can enter Page title starting with **

Output: User can see specific Page Title Detail

Precondition:

User visits the web application and is on the home page.

User has once searched for something on the application and is on the result page.

Post Condition:

Web application redirects user to the Wikipedia Title page with highest relevance from the result set.

Exception:

I. Query term could not be found in the corpus
II. Null or Special Character (Except query with prefix '**') Query term search.

Chapter 3

Design

1) Use Case Diagram

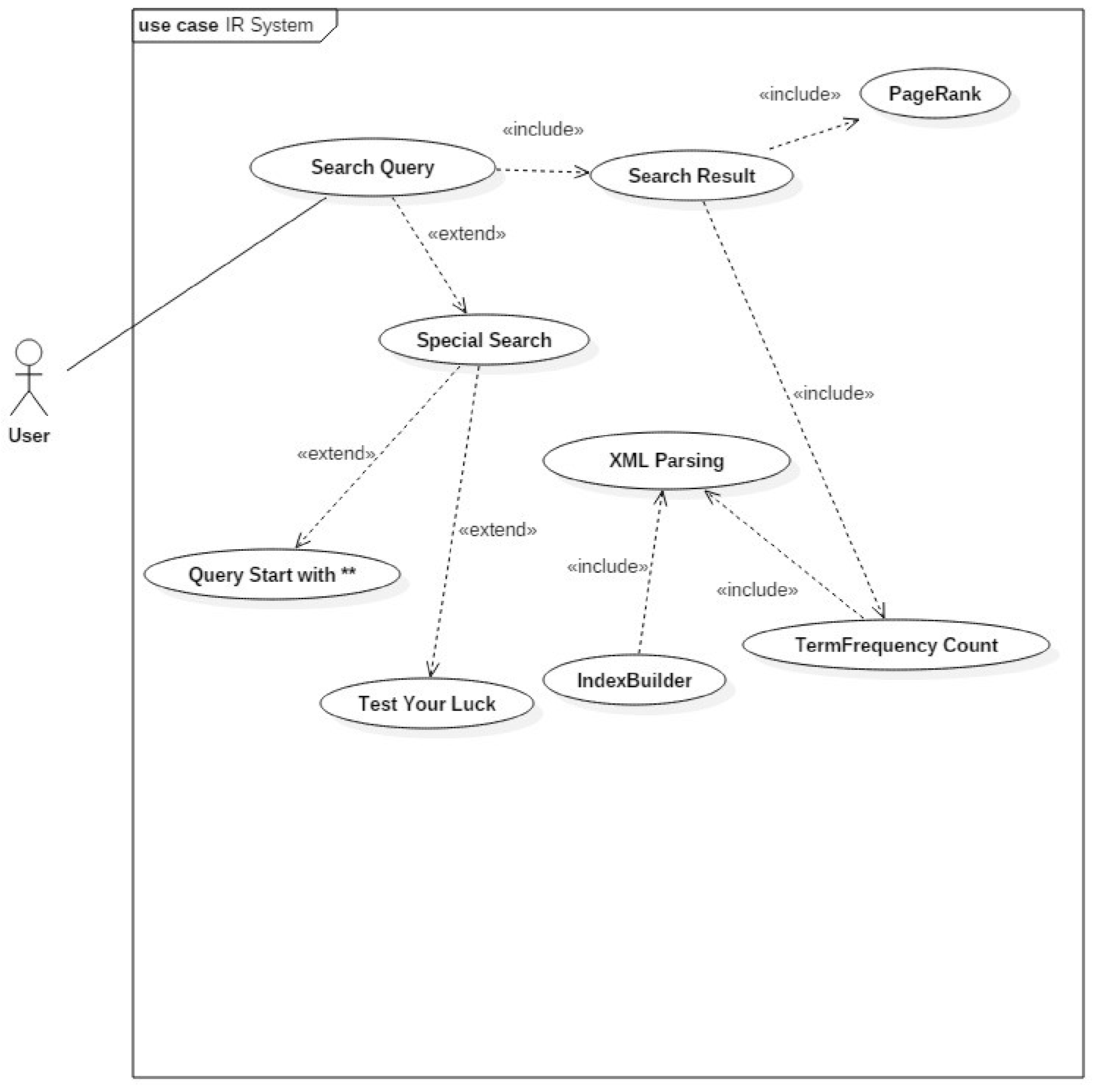


fig 3.1 use case diagram for IR System

2) Sequence Diagram

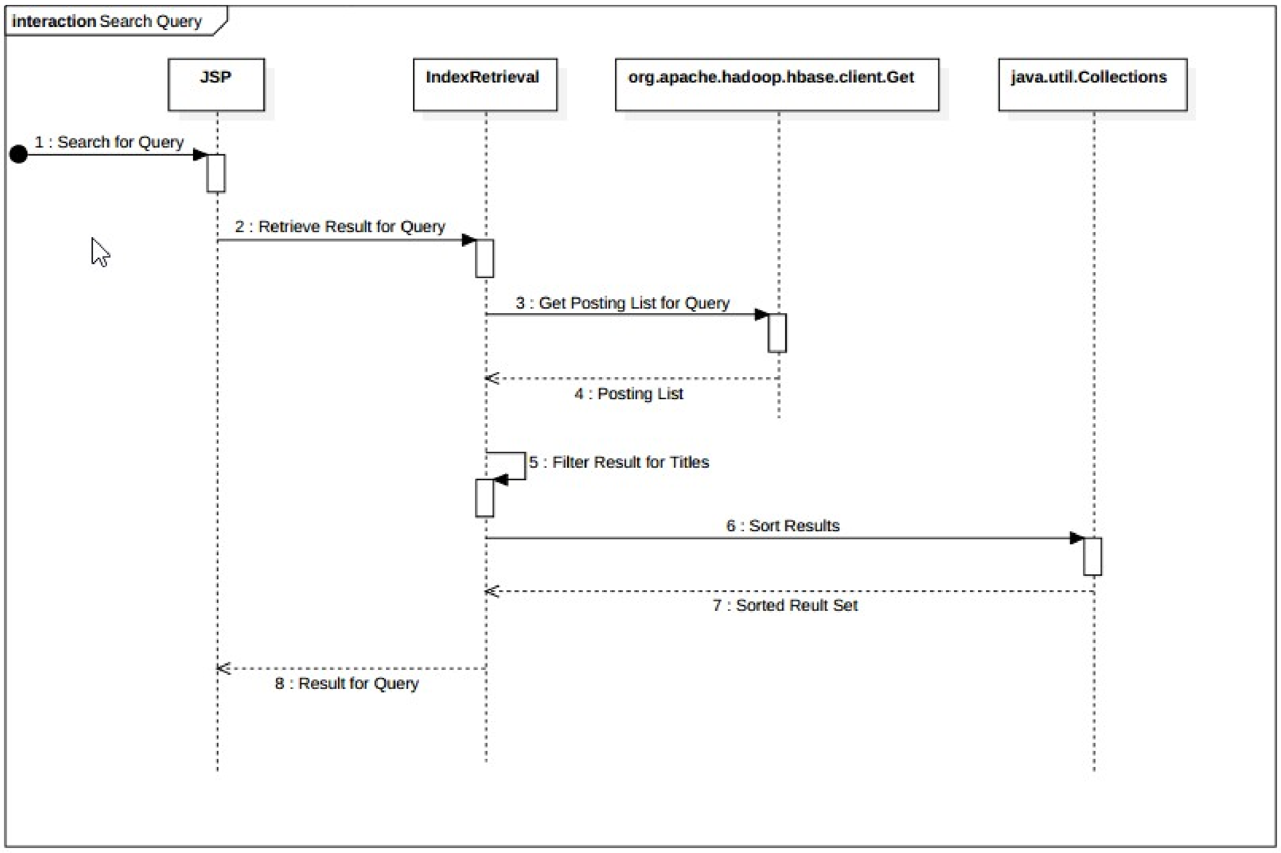


fig 3.2 sequence diagram for search query use case

3) Activity Diagram

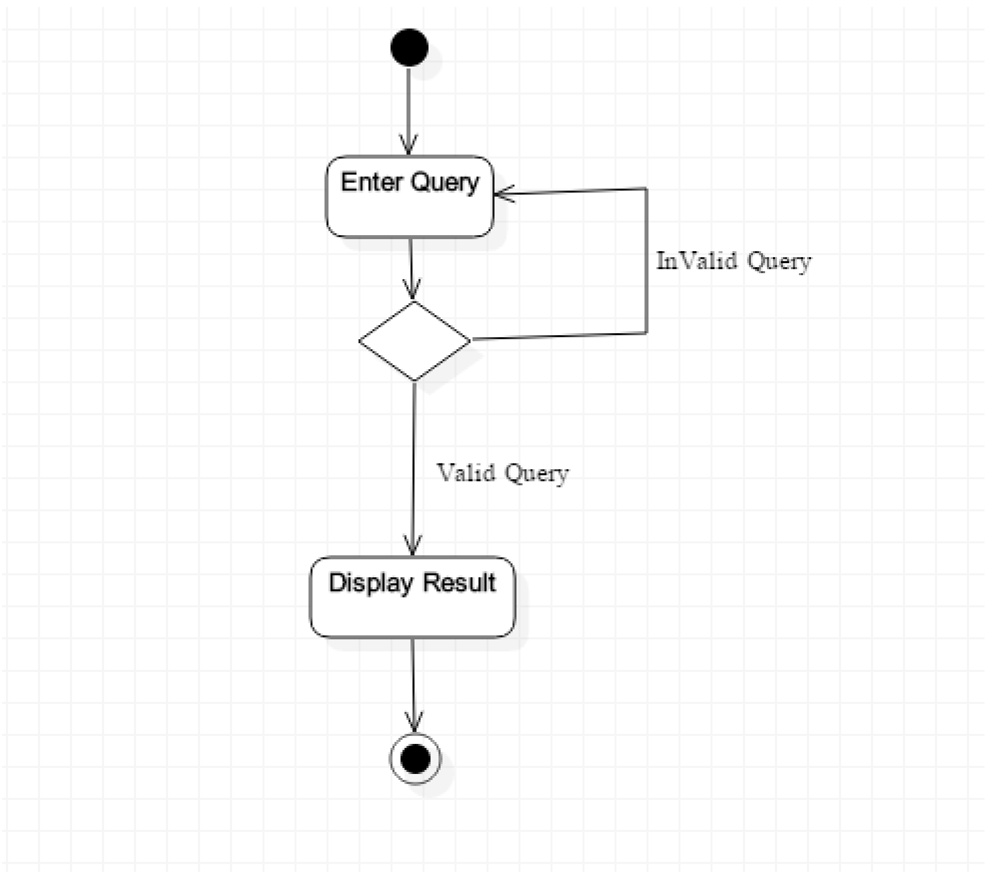


fig 3.3 activity diagram for simple querying

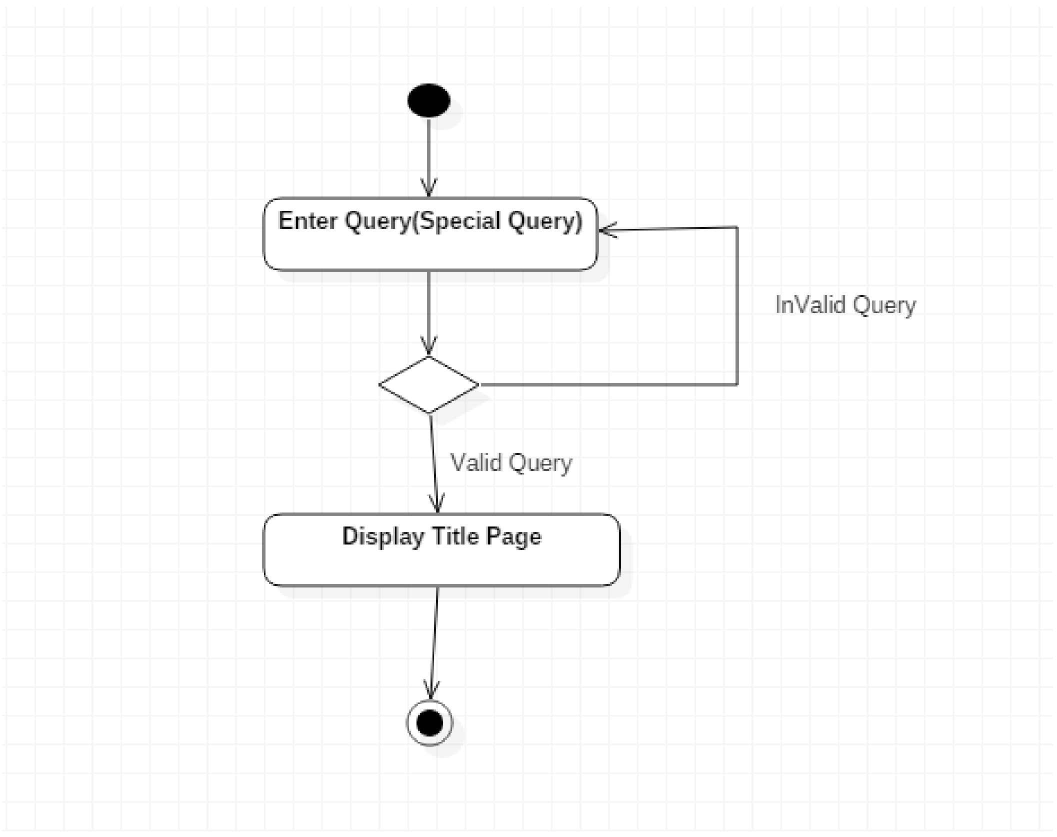


fig 3.4 activity diagram for special search

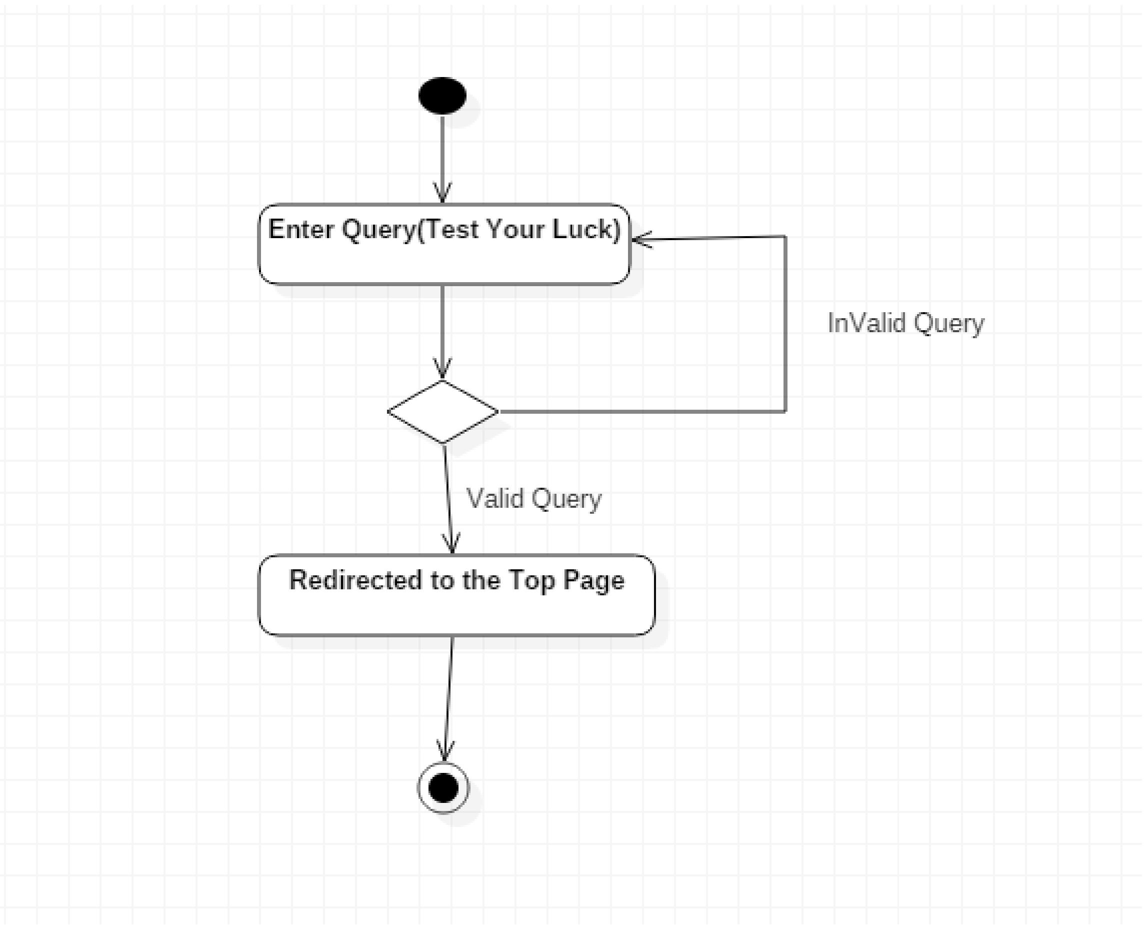


fig 3.5 activity diagram for test your luck feature

Chapter 4

Implementation Details

4.1) Implementation of Inverted Index

We have wikipedia data in XML format. We parsed xml using Mapreduce code and got required format output then we moved ahead.

XML Wikipedia document snippet:

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.10/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.mediawiki.org/xml/export-0.10/
http://www.mediawiki.org/xml/export-0.10.xsd" version="0.10" xml:lang="en">
  <siteinfo>
    <sitename>Wikipedia</sitename>
    <dbname>enwiki</dbname>
    <base>https://en.wikipedia.org/wiki/Main_Page</base>
    <generator>MediaWiki 1.27.0-wmf.7</generator>
    <case>first-letter</case>
    <namespaces>
      <namespace key="-2" case="first-letter">Media</namespace>
       <page>
    <contributor>
        <username>Kethrus</username>
        <id>22883165</id>
    </contributor>
   <title>AccessibleComputing</title>
 <text xml:space="preserve">{{Wiktionary|argument}}

In philosophy and logic, an '''[[argument]]''' is an attempt to persuade someone of
something, or give evidence or reasons for accepting a particular conclusion.
'''Argument''' may also refer to:
{{TOC right}}
{{anchor|Mathematics}}
........
</text>
</page>
```

Construction of Index:

Above is our example of wikipedia document format. From that document we we did was we parsed the xml file and build inverse indexed.

Our output of this document should be in below format

word, docId, frequency of that word in that perticular document

“India”, <8881,3>, <233,4>, <339,3>, <2007,2>, <7499,3>...

this is called postinglist of word.

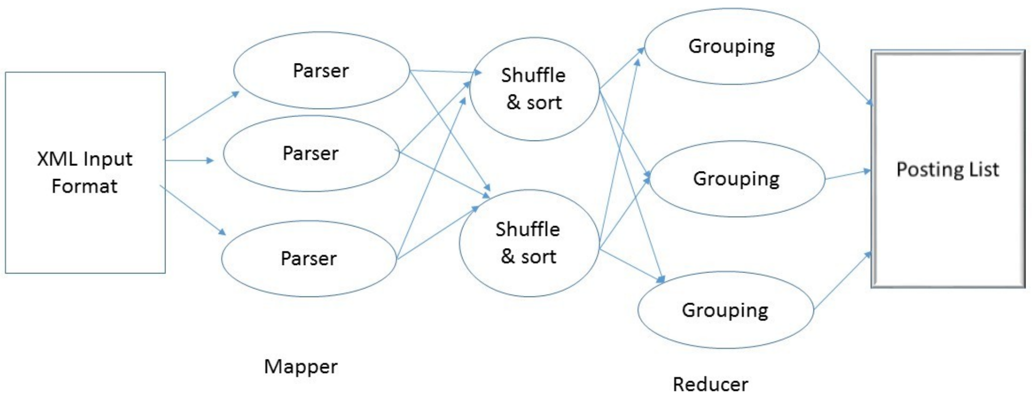


Fig 4.1 Typical Mapreduce Program Flow for creating Index

The inverted index technology has been widely used in information retrieval systems for searching text data, and the most well known implementation is the Apache Lucene library However, most existing Lucene-based systems, such as Solr maintain index data with files, and thus do not have a natural integration with HBase. Therefore, we propose a novel framework that can build inverted indices for text data in HBase, and store inverted index data directly as HBase tables. We call this framework IndexedHBase. Leveraging the distributed architecture of HBase, IndexedHBase can achieve reliable and scalable index data storage, as well as high performance for index data access. Moreover, by choosing proper searching strategies based on inverted indices, IndexedHBase can improve searching performance by several orders of magnitude, and therefore supports interactive analysis very well.

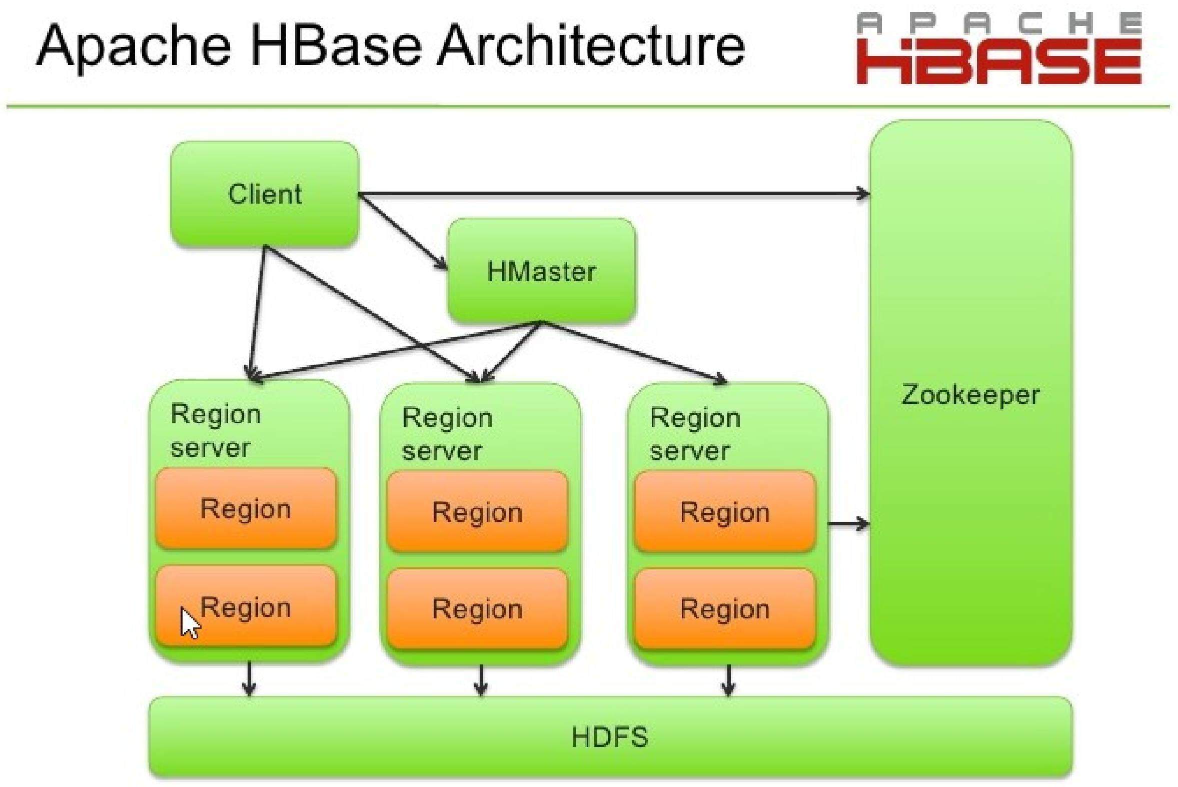


Fig 4.2 Hbase Architectur

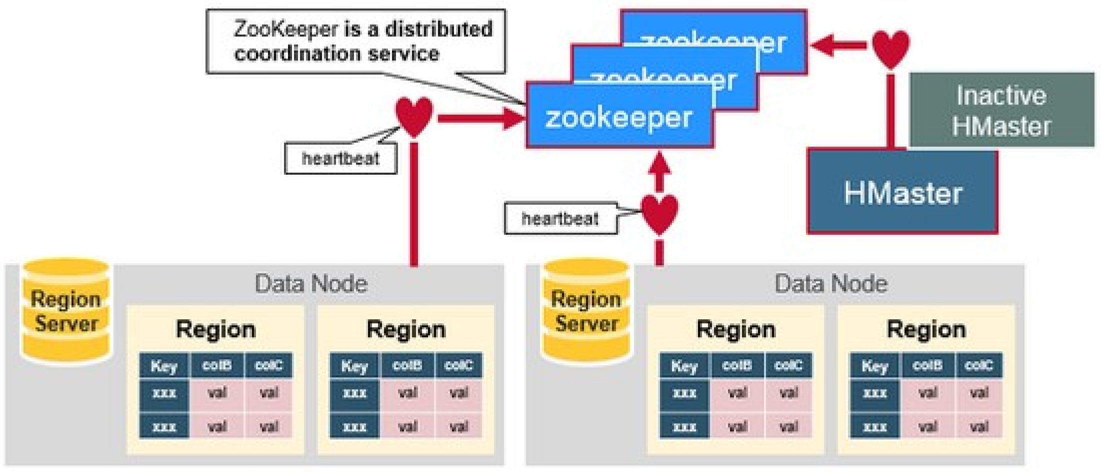


fig 4.3 Hbase working

Table 4.1 Hbase Table Architecture for Inverted Indexing

Table: word_docid_frequency				
Column Family: postinglist1				
rowkey	881(docID)	233(docID)	339(docID)	...(Other docIDs)
India	3	4	3	...

In order to store text data and index data in HBase tables, propertable schemas are needed. Figure 6 illustrates major table schemas in IndexedHBase. This table is named as “word_docid_frequency” and this table has only one column family named as “postinglist1”.each row has number of columns depends on how many docmuments containing that perticular word.

Using this table to store index data brings the following advantages to IndexedHBase:
(1) Since rows are sorted by row keys in HBase, it is easy to do a complete range scan of terms in index tables. This can be very helpful for evaluating queries containing wild characters, such as "ab*". Besides, since the qualifiers (document IDs) in each row are also sorted internally by HBase, it is easy to merge the index records for multiple terms.

(2) Since HBase is designed for efficient random access to cell data in tables, IndexedHBase can support very fast real-time document updates. The insertion, update, or deletion of a document only involves random write operations to a limited number of rows in these tables, and has little impact on the overall system performance, because HBase supports atomic operations at row level.

(3) Based on the original support for Hadoop MapReduce in HBase, we can develop efficient parallel algorithms for building inverted indices.

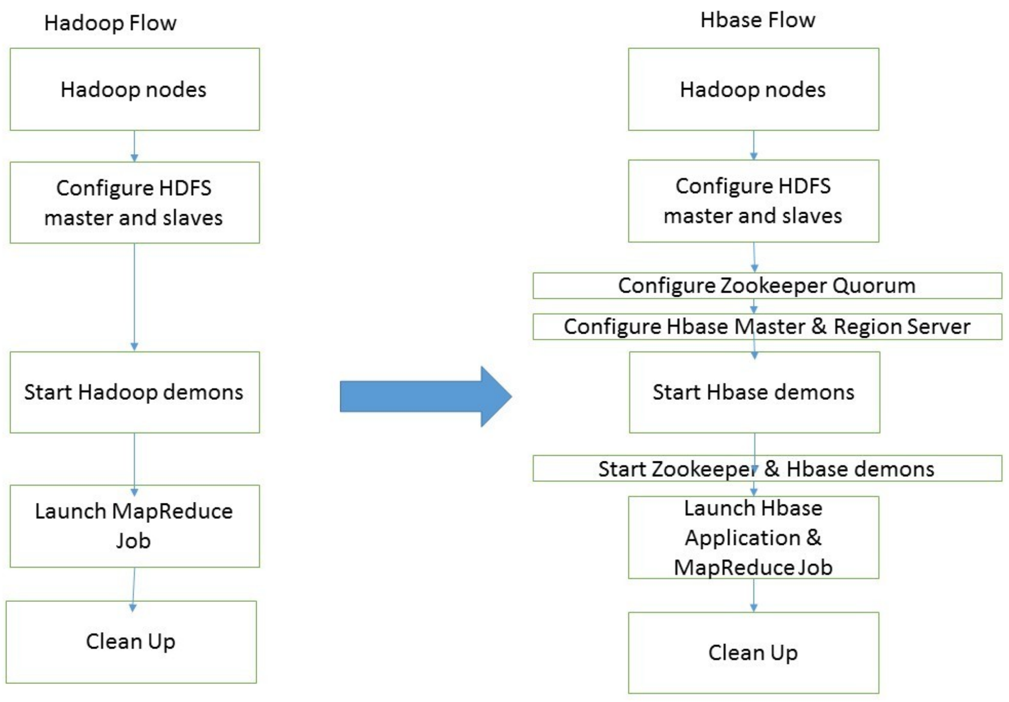

Hadoop Flow
Hadoop nodes
Configure HDFS master and slaves
Start Hadoop demons
Launch MapReduce Job
Clean Up
Hbase Flow
Hadoop nodes
Configure HDFS master and slaves
Configure Zookeeper Quorum
Configure Hbase Master & Region Server
Start Hbase demons
Start Zookeeper & Hbase demons
Launch Hbase Application & MapReduce Job
Clean Up

Fig.4.4 Hadoop and Hbase Flowchart

After running index creation MapReduce Code HBase and Hadoop will be running and available for data storage and MapReduce job execution. The second task is a MapReduce program that loads data from the Inverted Index data set to Word_docid_frequency in HBase.We use the HBase bulk loading strategy to finish this process, because this is the most efficient way to load data into HBase tables in large batches.

The whole process consists of the following two steps: (1)Run a MapReduce program to scan Word_docid_frequency table, build inverted index for all documents, and write index data to HDFS files in the HFile format, which is the file format HBase internally uses to store table data in HDFS.

(2) Import the HDFS files generated in step (1) to Word_docid_frequency table using the "CompleteBulkLoad" tool provided by Hbase. Step (2) normally finishes very fast (in seconds), and the major work is done in step (1).

Function Details:

MapReduce job: FrequencyInvertedIndex

This Job takes the path to the corpus as input, parse them into tokens and creates Inverted Index with tokens to document frequency.

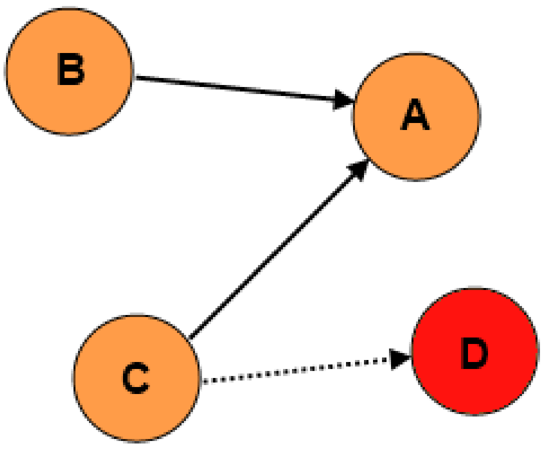
4.2) Implementation of PageRank Algorithm

"PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites."

PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyper linked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element *E* is referred to as the *PageRank of E* and denoted by $PR(E)$.

example:

In this example I will use 4 pages: A, B, C and D an non-existing page. This is a page that has not been created yet, but is being links to from C. In wikipedia you recognize those pages as red and underlined. The links between the pages are as follows:



Rank of A is highest, because it will get points from B and C.

PageRank of page A = 'share' of the PageRank of the pages linking to A.

The formula of calculating the points is as following:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

The formula can be simplified to this:

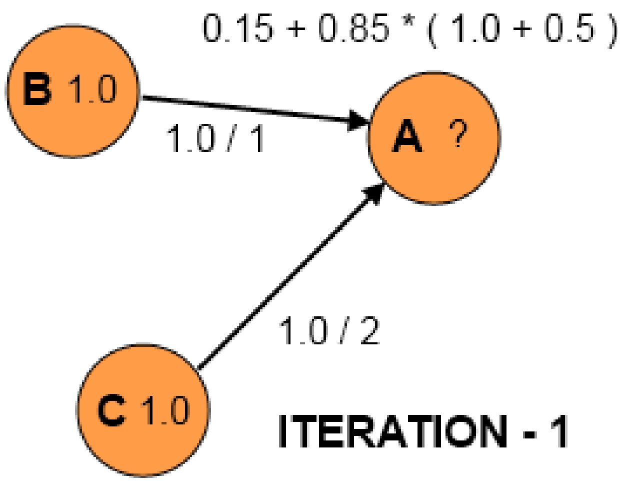
PR(A) = (1-d) + d(PR(B) / Cout(B) + ... + PR(C) / Cout(C))

The d in the formula is the damping factor to simulate 'a random surfer' and is usually set to 0.85.

If you apply the formula to our example:

*PageRank of A = 0.15 + 0.85 * (PageRank(B)/outgoing links(B) + PageRank(...)/outgoing link(...))*

Calculation of A with initial ranking 1.0 per page:



If we use the initial rank value 1.0 for A, B and C we would have the following output:

I have skipped page D in the result, because it is not an existing page.

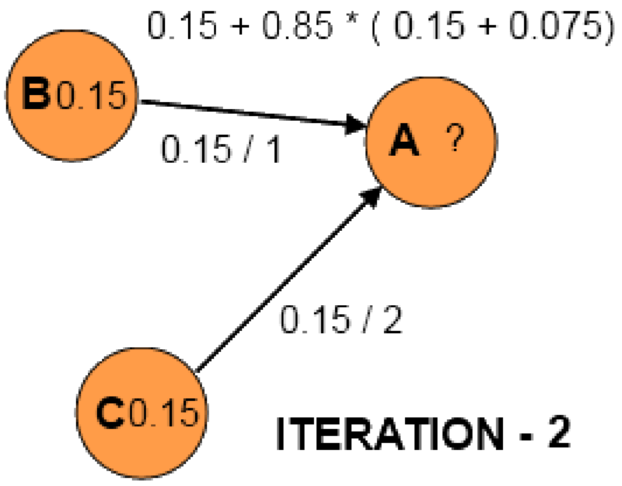
A: 1.425

B: 0.15

C: 0.15

Calculation of A with ranking from ITERATION-1:

If we use these ranks



as input and calculate it again:

A: 0.34125

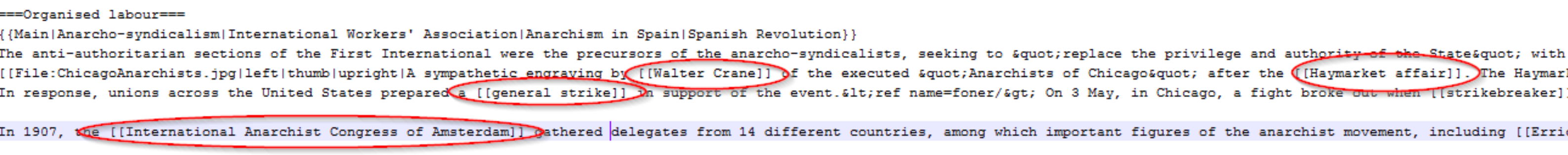
B: 0.15

C: 0.15

We see that the page rank of page A is reduced. The PageRank is based on previous calculations and will get more accurate after more runs. You can add new pages, new links in the future and calculate the new rankings. This is one of the tools which search engines use to create there index. We had done this with a set of wikipedia pages.

- **How we decided which pages are connected with which other Pages?**

Below image is one snapshot of Wikipedia Article named “Anarchism”.Wikipedia put other connected wikipedia article in double square bracket **“[[other page title]]”**.As you can see red marked circle that all are other connected pages with this page.



```
===Organised labour===
{{Main|Anarcho-syndicalism|International Workers' Association|Anarchism in Spain|Spanish Revolution}}
The anti-authoritarian sections of the First International were the precursors of the anarcho-syndicalists, seeking to &quot;replace the privilege and authority of the State&quot; with
[[File:ChicagoAnarchists.jpg|left|thumb|upright|A sympathetic engraving by [[Walter Crane]] of the executed &quot;Anarchists of Chicago&quot; after the [[Haymarket affair]]. The Haymar
In response, unions across the United States prepared a [[general strike]] in support of the event.&lt;ref name=foner/&gt; On 3 May, in Chicago, a fight broke out when [[strikebreaker]]

In 1907, the [[International Anarchist Congress of Amsterdam]] gathered delegates from 14 different countries, among which important figures of the anarchist movement, including [[Erric
```

Fig 4.5 snapshot of paragraph of one article

We parsed these file and get this all types of new connected page ids with current page id and based on that we implemented our algorithm.

For that we already implemented program for getting page id from page title as output like

page title,	page id
autism,	25

For easier retrieval we also implemented program for getting pagetitle from page id as ouptput like

page id,	page title
25,	autism

Function Details:

This Implementation is divided into series of MapReduce jobs.

i. MapReduce job: TitleIDTitle
This job takes input as path to the corpus and parses them to output an index which maps a TitleID to Title from the corpus.

ii. MapReduce job: OutGoingLinks
This job takes input as path to the corpus and uses the Index created in the previous step to create TitleID to OutGoingTitlesID<List>.

iii. MapReduce job: PageRank
This job takes input as the file created in the previous step to calculate as assign PageRank score to each of the Title pages.

4.3) Implementation of TF Algorithm

tf–idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.It is often used as a weighting factor in information retrieval and text mining. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. tf–idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

How to Compute:

Typically, the tf- weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document.

- **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

weighting scheme	**TF weight**
binary	$0, 1$
raw frequency	$f_{t,d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Fig 4.6 Variants of TF weight

Example:

Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03.

Now, assume we have 10 million documents and the word cat appears in one thousand of these.

How We Implemented?

For TF We have Frequency of each word for each document

so we simply took that value and put in formula of TF:

We used below formula for counting TF Value:(log normalization)

$$TF = 1+Math.log_{10}(Frequency\ of\ Word)$$

Function Details:

This Implementation is divided into series of MapReduce jobs.

i. MapReduce job: TitleIDTitle
This job takes input as path to the Inverted Index and parses them to output an index which maps a TitleID to words in it.

ii. MapReduce job: IDF
This job takes input as the file created in the previous step to calculate as assign InverseDocumentFrequency score to each of the Title pages.

4.4) How We implemented Final Retrieval Result?

Whenever user enter his query we look for that query into our postinglist from that we will get word's Term Frequency and Word's Page ID.We can get Page Rank Of the Page from Page ID.

We calculated PageRank*TermFrequency(TF) and sort that Document List and show to the user.

In Special Search Case we first check that query is starting with ** or not. if it is then we tried to find that exact title from our Hbase Table and if it is match then we show that Document to user

e.g user wants to try “database ” page title then he should try ****database** then go.

- We used **OOZIE** for Automation of all these jobs
- **What is OOZIE?**

Oozie is a workflow scheduler system to manage Apache Hadoop jobs.

What Oozie Does?

Apache Oozie is a Java Web application used to schedule Apache Hadoop jobs. Oozie combines multiple jobs sequentially into one logical unit of work. It is integrated with the Hadoop stack, with YARN as its architectural center, and supports Hadoop jobs for Apache MapReduce, Apache Pig, Apache Hive, and Apache Sqoop. Oozie can also schedule jobs specific to a system, like Java programs or shell scripts.

Apache Oozie is a tool for Hadoop operations that allows cluster administrators to build complex data transformations out of multiple component tasks. This provides greater control over jobs and also makes it easier to repeat those jobs at predetermined intervals. At its core, Oozie helps administrators derive more value from Hadoop.

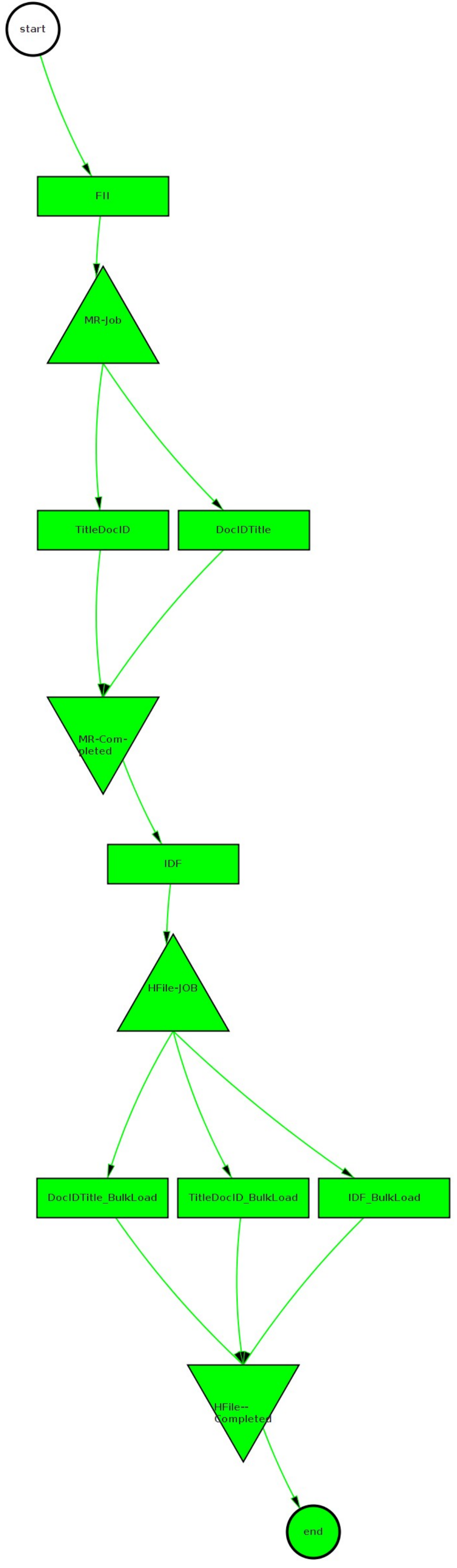
There are two basic types of Oozie jobs:

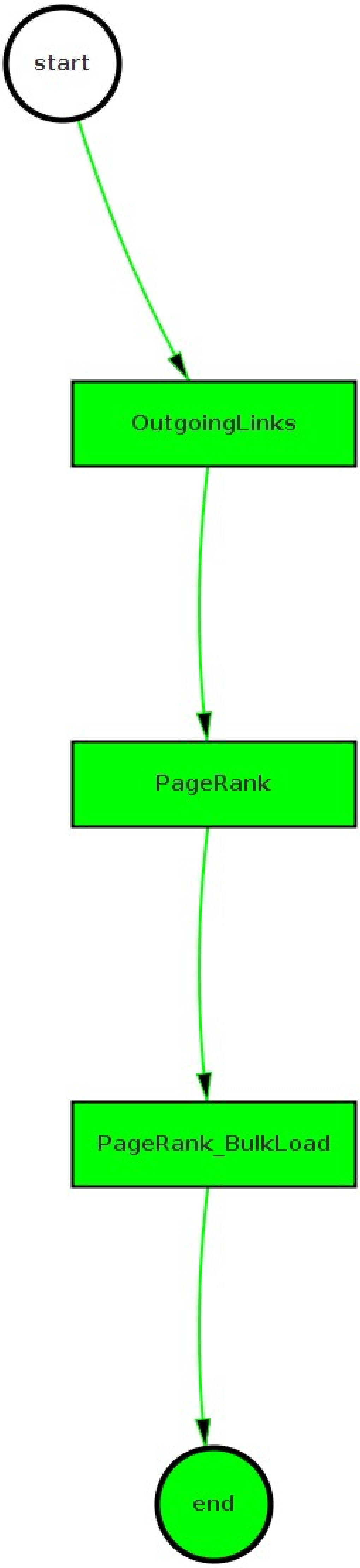
•**Oozie Workflow** jobs are Directed Acyclical Graphs (DAGs), specifying a sequence of actions to execute. The Workflow job has to wait

•**Oozie Coordinator** jobs are recurrent Oozie Workflow jobs that are triggered by time and data availability.

Oozie Bundle provides a way to package multiple coordinator and workflow jobs and to manage the lifecycle of those jobs.

Here is Our Job DAGs(Directed Ascyclic Graph)


start
FII
MR-Job
TitleDocID
DocIDTitle
MR-Com-
pleted
IDF
HFile-JOB
DocIDTitle_BulkLoad
TitleDocID_BulkLoad
IDF_BulkLoad
HFile--
Completed
end


start
OutgoingLinks
PageRank
PageRank_BulkLoad
end

Chapter 5

Testing

I) Frequency Inverse Index(FII) Testing:

For FII Testing we took small XML file representing actual input(Wikpedia dump).

Input file:

```
<page>
        <id>1</id>
        <title>Testing XML InputFormat and XML Parsing</title>
        <text>
                this is some RANDOM text, with some numbers 1,3, 8,90  , 9009s, with is
is is is is is is
        </text>
</page>
<page>
        <id>2</id>
        <title>Testing XMLIF and XMLP</title>
        <text>
                this is some also random random     random   text with more spaces,
numbers 1,3, 8,90  , 9009s,
==PLOT==THIS KIND of TEXT NOT ALLOWED
| this also......
|
{{fojdsfogss###Ihaosfifeh**(Fdahfai}}
|
|
```

==REFERENCE=

</text>

</page>

and we expected output is

also,2,1.0

is,2,1.0,1,1.90

more,2,1.0

number,2,1.0,1,1.0

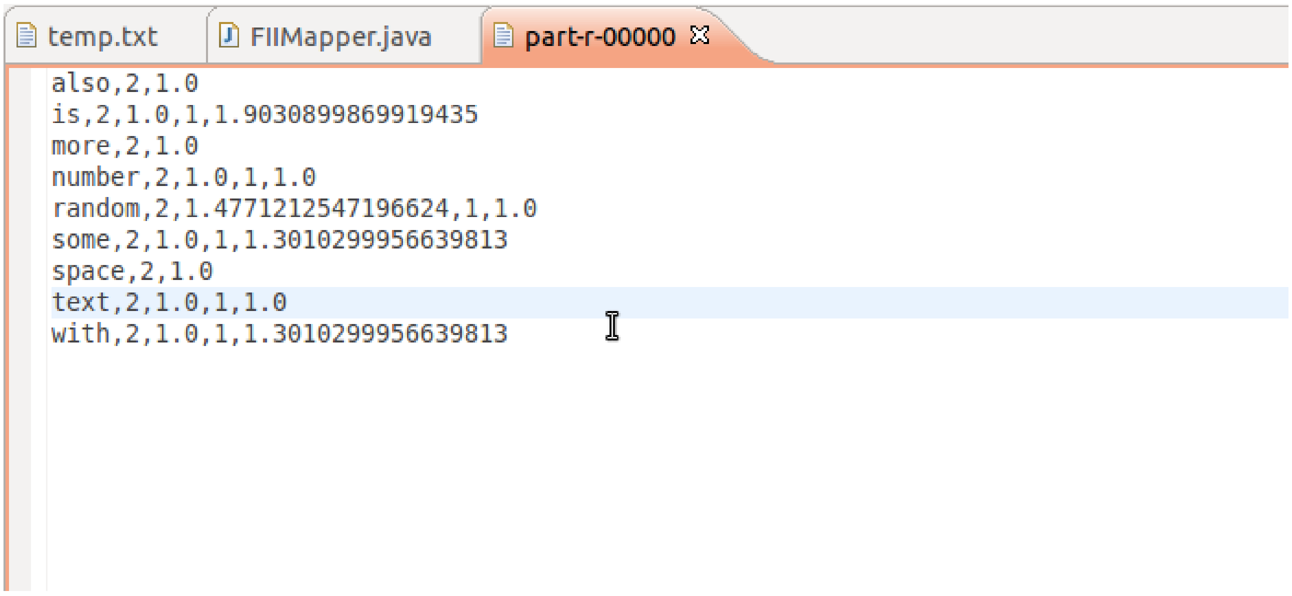
random,2,1.47

some,2,1.0,1,1.30

space,2,1.0

text,2,1.0,1,1.0

Output we got by executing FII MapReduce Program



```
also,2,1.0
is,2,1.0,1,1.9030899869919435
more,2,1.0
number,2,1.0,1,1.0
random,2,1.4771212547196624,1,1.0
some,2,1.0,1,1.3010299956639813
space,2,1.0
text,2,1.0,1,1.0
with,2,1.0,1,1.3010299956639813
```

fig 5.1 snapshot of output for inverse index

ii) PageRank Testing

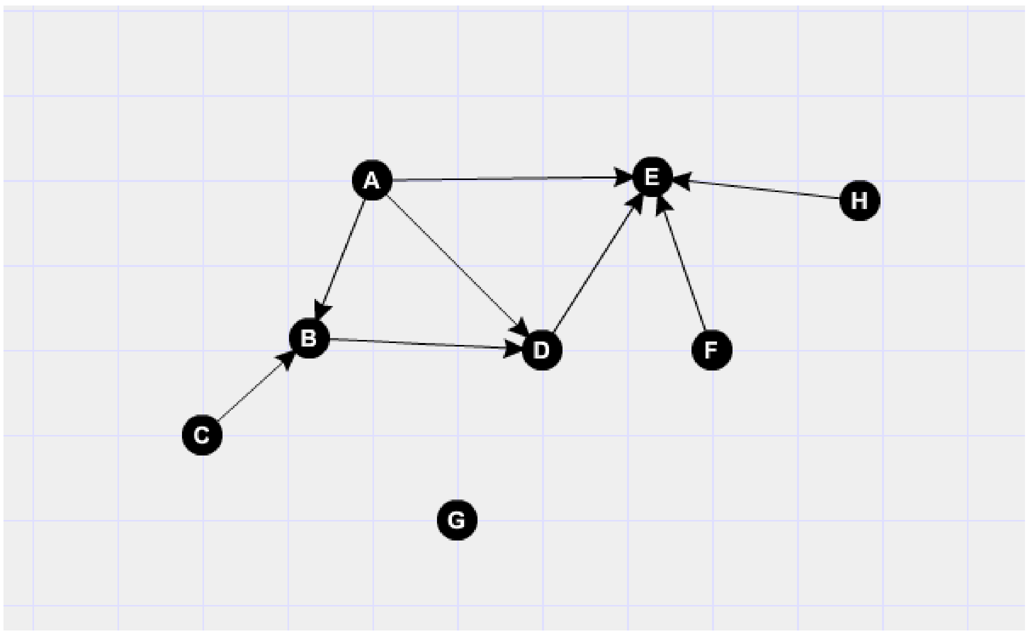


fig 5.2 small graph of which pages are conneced to which other pages

Hence input as csv file representing graph is this

1,2,4,5

2,4

3,2,4

4,5

5,

6,5

7,

8,5

and expected output is

1,0.25

2,1.08

3,0.33

4,1.58

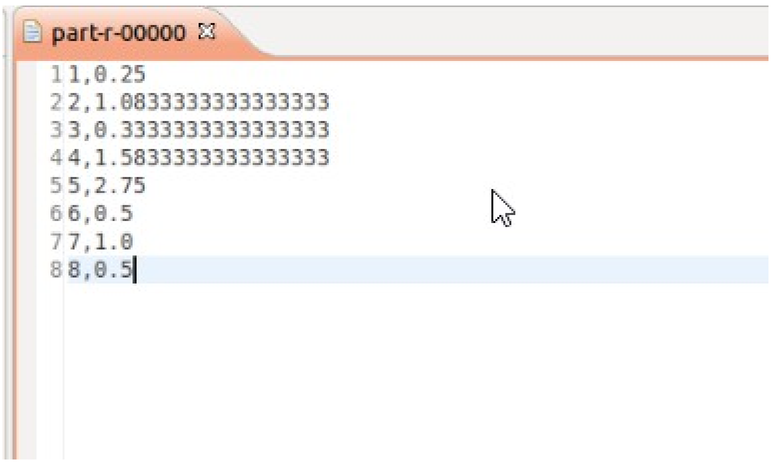
5,2.75

6,0.5

7,1.0

8,0.5

We obtained output as



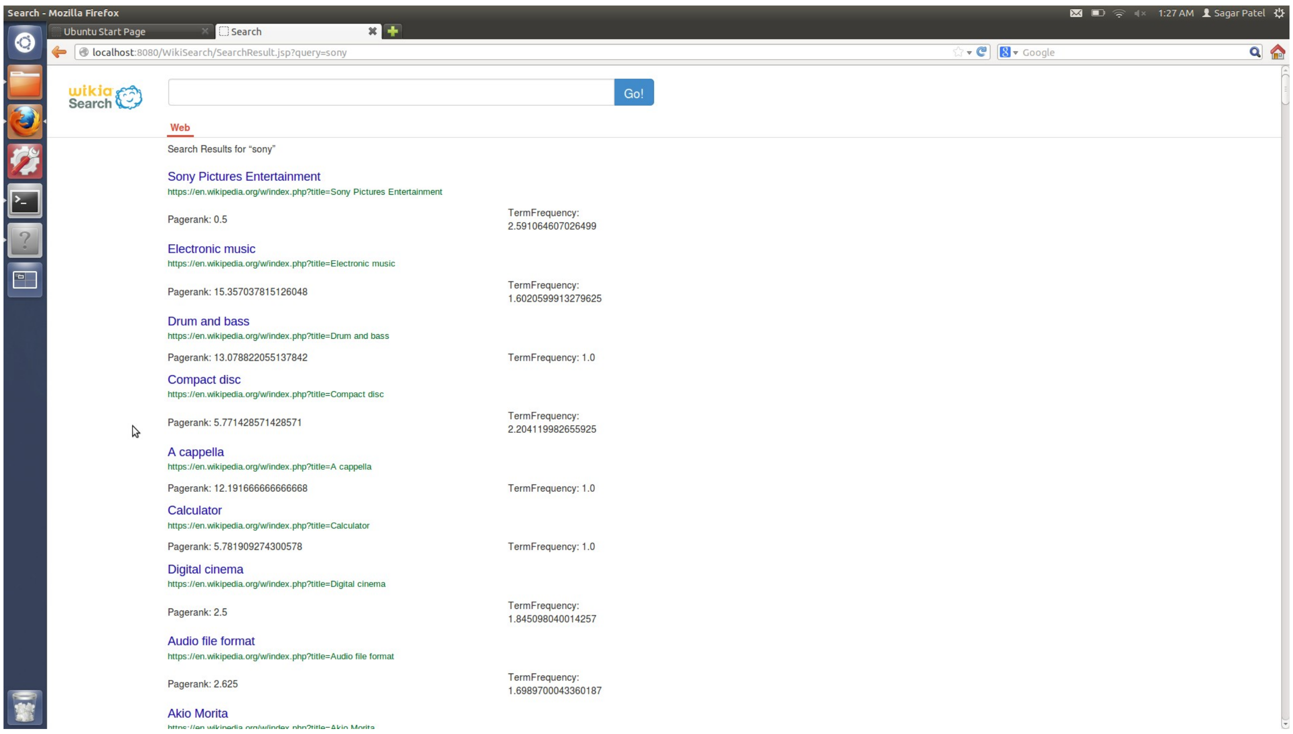
part-r-00000

```
1,0.25
2,1.0833333333333333
3,0.3333333333333333
4,1.5833333333333333
5,2.75
6,0.5
7,1.0
8,0.5
```

fig 5.3 snapshot of output of PageRank

Chapter 6

Screen Shot of Information Retrieval System



Search - Mozilla Firefox

Ubuntu Start Page | Search

localhost:8080/WikiSearch/SearchResult.jsp?query=sony

Search Results for “sony”

Sony Pictures Entertainment
https://en.wikipedia.org/w/index.php?title=Sony Pictures Entertainment
Pagerank: 0.5 TermFrequency: 2.591064607026499

Electronic music
https://en.wikipedia.org/w/index.php?title=Electronic music
Pagerank: 15.357037815126048 TermFrequency: 1.6020599913279625

Drum and bass
https://en.wikipedia.org/w/index.php?title=Drum and bass
Pagerank: 13.078822055137842 TermFrequency: 1.0

Compact disc
https://en.wikipedia.org/w/index.php?title=Compact disc
Pagerank: 5.771428571428571 TermFrequency: 2.204119982655925

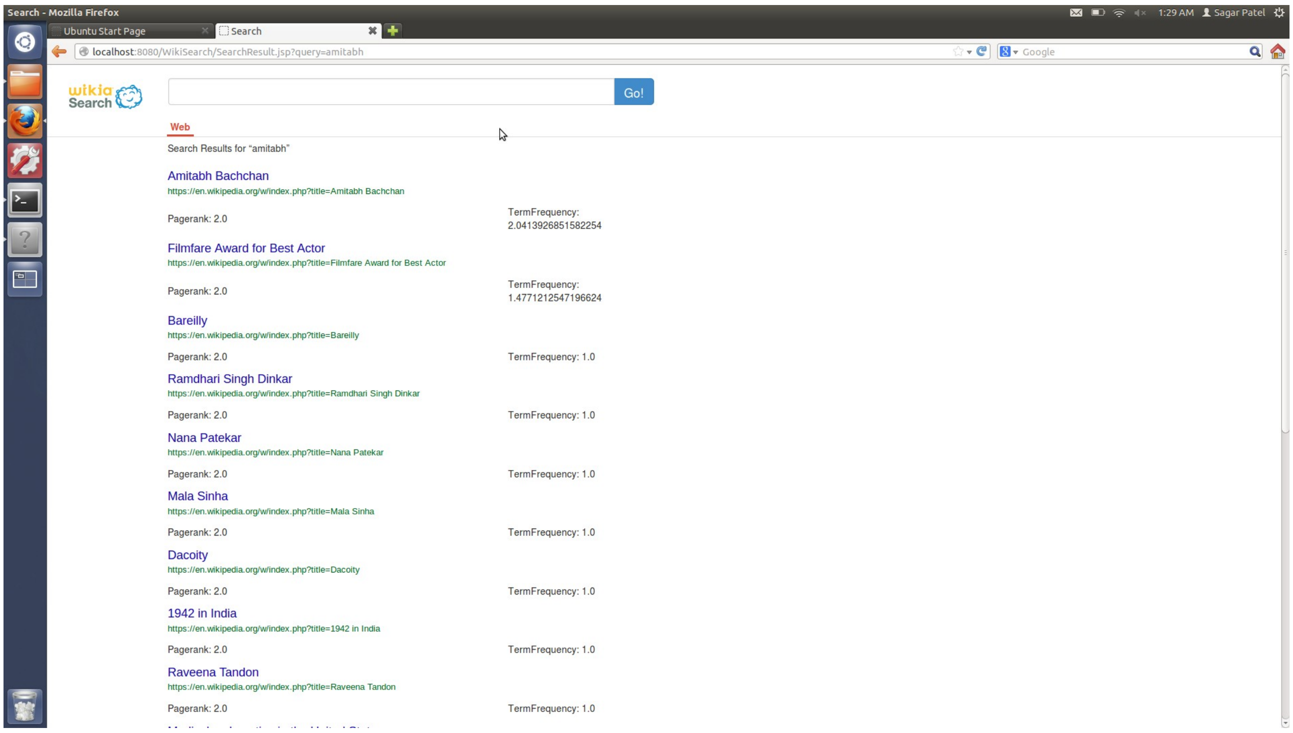
A cappella
https://en.wikipedia.org/w/index.php?title=A cappella
Pagerank: 12.191666666666668 TermFrequency: 1.0

Calculator
https://en.wikipedia.org/w/index.php?title=Calculator
Pagerank: 5.781909274300578 TermFrequency: 1.0

Digital cinema
https://en.wikipedia.org/w/index.php?title=Digital cinema
Pagerank: 2.5 TermFrequency: 1.845098040014257

Audio file format
https://en.wikipedia.org/w/index.php?title=Audio file format
Pagerank: 2.625 TermFrequency: 1.6989700043360187

Akio Morita
https://en.wikipedia.org/w/index.php?title=Akio Morita



Search Results for “amitabh”

Amitabh Bachchan
https://en.wikipedia.org/w/index.php?title=Amitabh Bachchan
Pagerank: 2.0 TermFrequency: 2.0413926851582254

Filmfare Award for Best Actor
https://en.wikipedia.org/w/index.php?title=Filmfare Award for Best Actor
Pagerank: 2.0 TermFrequency: 1.4771212547196624

Bareilly
https://en.wikipedia.org/w/index.php?title=Bareilly
Pagerank: 2.0 TermFrequency: 1.0

Ramdhari Singh Dinkar
https://en.wikipedia.org/w/index.php?title=Ramdhari Singh Dinkar
Pagerank: 2.0 TermFrequency: 1.0

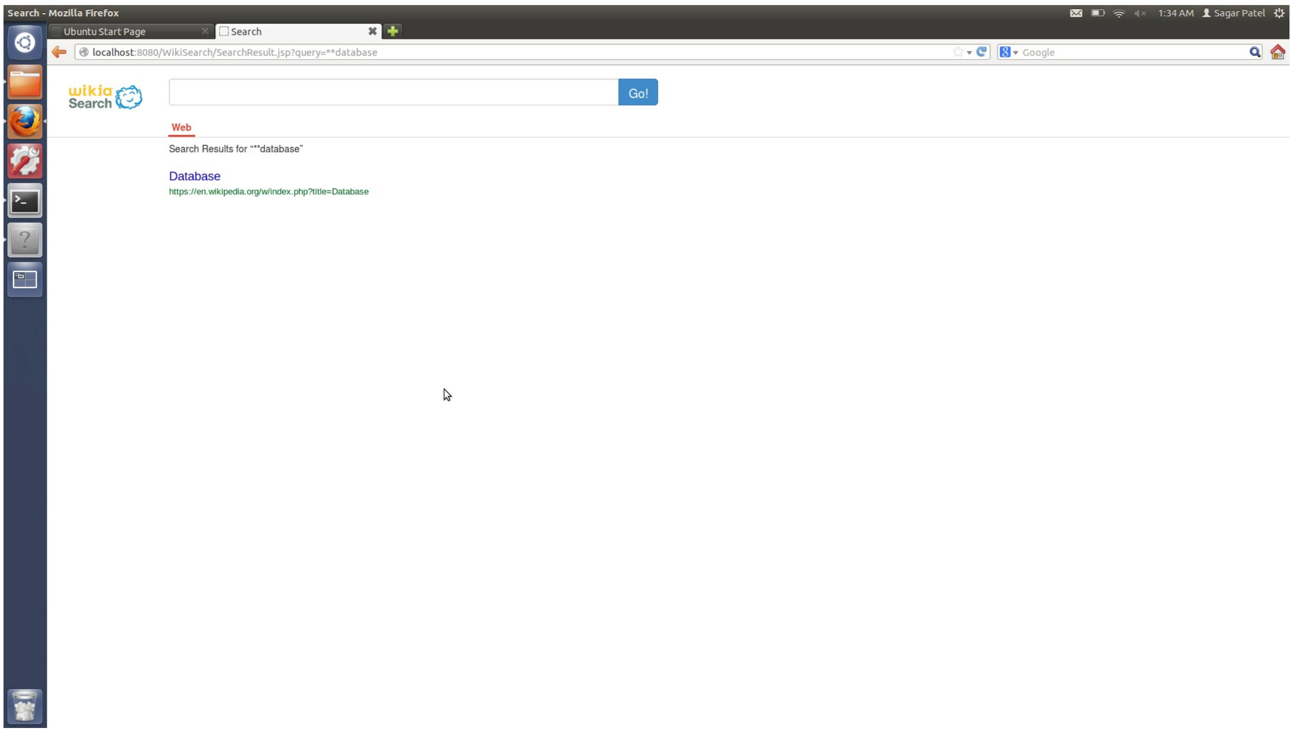
Nana Patekar
https://en.wikipedia.org/w/index.php?title=Nana Patekar
Pagerank: 2.0 TermFrequency: 1.0

Mala Sinha
https://en.wikipedia.org/w/index.php?title=Mala Sinha
Pagerank: 2.0 TermFrequency: 1.0

Dacoity
https://en.wikipedia.org/w/index.php?title=Dacoity
Pagerank: 2.0 TermFrequency: 1.0

1942 in India
https://en.wikipedia.org/w/index.php?title=1942 in India
Pagerank: 2.0 TermFrequency: 1.0

Raveena Tandon
https://en.wikipedia.org/w/index.php?title=Raveena Tandon
Pagerank: 2.0 TermFrequency: 1.0



Search Results for “**database”

Database
https://en.wikipedia.org/w/index.php?title=Database

Chapter 7

Conclusion

- We implemented the functionality, Single Word Query Search with reasonable relevance for the corpus (Wikipedia Dataset).
- We implemented the functionality, Special Title Search for the corpus.
- We implemented Inverted Index, PageRank Scores, Term Frequency-Inverse Document Frequency (TF-IDF) scores for various tokens from the corpus.

We used Wikipedia Dataset of size almost **450 MB** which contains **59,968** English language Articles.

- ✔ Total Words: **4,50,544**
- ✔ Highest Frequency Word : **time (31,496)**
- ✔ Highest Page Rank/Most Connected Page: **91.82 (Page ID: 6416 Page Title: Impact crater)**
- ✔ Lowest Page Rank/Least Connected Page : **0.007(Page ID: 956486 Page Title: List of Volkswagen Group petrol engines)**

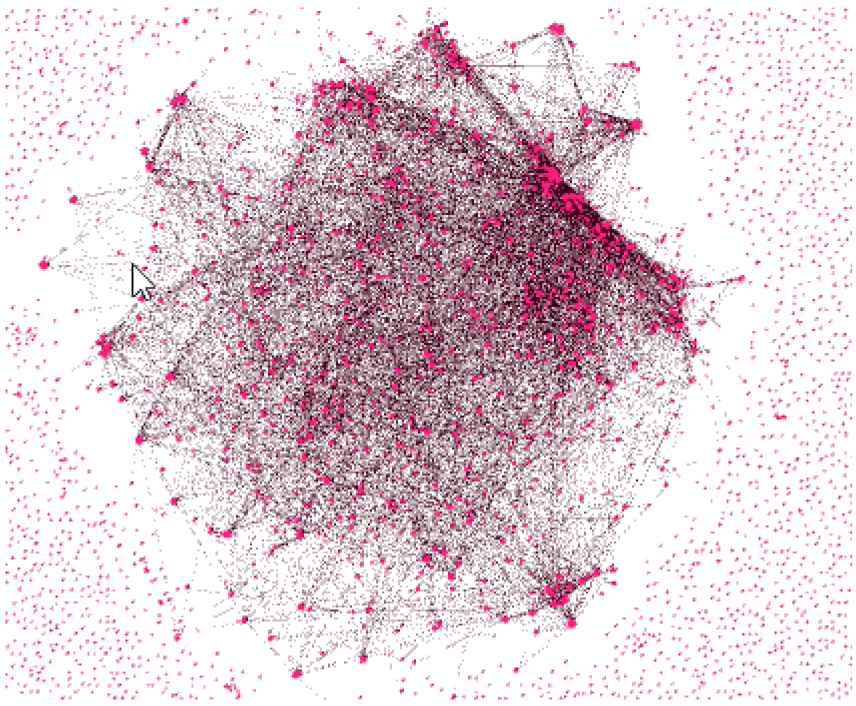


fig 7.1 visualization graph of how all pages are connected with each other

Above graph is created using Gephi Tool which is used for network analysis and visualization. That graph shows connection between all the pages which used for counting the Page Rank. That Graph contains **59968 Nodes and 87814 Edges.**

Limitation and Future Extension:

1. No support for Phrase Query or Wild card query.

2. Lack of high quality relevancy and the ranking of the result set.

3. Large size of index, index compression not used.

4. Relevancy could have been improved by the use of meta information from the pages.

5. Same words in different form(verb, with some suffix, prefix) have different results, as the tokenized words were not normalized.

Bibliography

[1] Introduction to Information Retrieval By Christopher D. Manning(Author), Prabhakar Raghavan (Author), Hinrich Schütze(Author)

[2] Hadoop: The Definitive Guide - 3rd Edition By Tom White (Author)

[3] Page Rank Reference - http://blog.xebia.com/wiki-pagerank-with-hadoop/

[4] Page Rank Wikipedia - https://en.wikipedia.org/wiki/PageRank

[5] Scalable Inverted Indexing on NoSQL Table Storage- http://dsc.soic.indiana.edu/publications/Scalable Inverted Indexing_v4B.pdf

[6] Wikipedia Data Parser Reference - https://bitbucket.org/axelclk/info.bliki.wiki/wiki/Home

[7] Hbase Bulk Load Reference - http://www.thecloudavenue.com/2013/04/bulk-loading-data-in-hbase.html