**PERCIPIENT AND COGNITIVE SEARCH ENGINE**

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Engineering

By

Ruchita Jadhav, 11-607

Madhura Jagtap, 11-608

Pradnesh Patil, 11-629

Under the Guidance of

Prof. Mithil Gharat



Department of Computer Engineering

Vidyalankar Institute of Technology

Wadala (E), Mumbai 400-037

University of Mumbai

2014-15

PROJECT REPORT APPROVAL FOR BACHELOR OF ENGINEERING

This Project Report entitled **Percipient and cognitive search engine** by **Ruchita Jadhav**, **Madhura Jagtap** and **Pradnesh Patil** is approved for the degree of **Bachelor of Engineering in Computer Engineering.**

Examiners

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Supervisors/ Guides

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Head of Department

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Principal

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date:

Place: Mumbai

I. DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of student Roll No. Signature

1. Ruchita Jadhav 11-607 \_\_\_\_\_\_\_\_\_\_
2. Madhura Jagtap 11-608 \_\_\_\_\_\_\_\_\_\_
3. Pradnesh Patil 11-629 \_\_\_\_\_\_\_\_\_\_

Date:

II. ABSTRACT

The project is an attempt to build a smarter search engine. Existing search engines use extensive crawlers and index these results based on the content, metadata and the sites that link to it. When a user enters a search query in an existing search engine, he is presented with a number of relevant links which are ranked by some proprietary algorithms (eg: PageRank) and a few sentences from that page containing the keyword.

Our project tries to ease the process of searching for a particular topic on the internet. The approach to search that has been noticed in students researching a particular topic is to open up numerous tabs and then go through it manually. Our project will serve as a dashboard for all the results. It will cache the heat web pages and then perform content search. It will summarize the content with emphasis on the keywords in the search query. It will also mine and summarize academic papers that are freely available. All these summaries will have links to their full pages.

If it is a current hot-topic, it will have presence on social media such as Twitter too. We will include an option to search Twitter feeds, which will display to the user prominent tweets (based on no. of RTs/favorites/likes) with a brief sentiment analysis on that feed.

II. ACKNOWLEDGEMENT

We take this opportunity to express our deepest gratitude towards our project guide Prof. Mithil Gharat, who has been the driving force behind this project and whose guidance and co-operation has been a source of inspiration for us.

We are very much thankful to our professors, colleagues and authors of various publications to which we have been referring to. We express our sincere appreciation and thanks to all those who have guided us directly or indirectly in our project. Also much needed moral support and encouragement was provided on numerous occasions by our whole division.

IV. TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| Sr. No. | Page Title | Page No. |
|  | Project Overview | 9 |
|  | Introduction and Motivation | 10 |
|  | * 1. Theory Behind the Project Concept | 11 |
|  | * 1. Problem Definition | 16 |
|  | * 1. Need for Project | 16 |
|  | * 1. Challenges | 17 |
|  | Analysis and Design | 18 |
|  | * 1. Software Development | 18 |
|  | * 1. Flow of Project | 19 |
|  | 3.2.1. Preliminary Study | 19 |
|  | 3.2.2. Feasibility Study | 23 |
|  | 3.2.3. Cost Analysis | 24 |
|  | 3.2.4. Process Model | 26 |
|  | 3.2.5. Data Flow Diagrams | 28 |
|  | * 1. UML Diagram | 29 |
|  | * 1. Technologies Used | 30 |
|  | * + 1. Hardware & Software Requirements | 31 |
|  | * + 1. Introduction to Programming Tools | 31 |
|  | Project Time & Task Distribution | 34 |
|  | * 1. Timeline Chart | 34 |
|  | Implementation | 35 |
|  | Test Cases | 43 |
|  | * 1. Graphical User Interface | 43 |
|  | * 1. Test Cases | 44 |
|  | Conclusion And Future Scope | 45 |
|  | Appendix A: User Manual | 47 |
|  | Appendix B: Classes and External Libraries,apis | 49 |
|  | Appendix C: Input & Output for Test Cases | 51 |
|  | Bibliography | 54 |

V. TABLE OF FIGURES

|  |  |  |
| --- | --- | --- |
| Fig. No. | Page Title | Page No. |
| 1.1 | High-level architecture of a standard Web crawler | 12 |
| 2.1 | Estimation table | 24 |
| 3.1 | Waterfall model | 26 |
| 3.2 | Dataflow diagram | 28 |
| 3.3 | Dataflow diagram | 28 |
| 3.4 | Usecase diagram | 29 |
| 3.5 | Component diagram | 29 |
| 3.6 | Hardware Requirements | 31 |
| 4.1 | Timeline Chart (For Semester 7) | 34 |
| 4.2 | Timeline Chart (For Semester 8) | 34 |
| 5.1 | Hadoop hbase cluster | 36 |
| 5.2 | Crawled content without the metatags. | 42 |
| 6.1 | GUI of the search engine | 43 |
| 6.2 | GUI of the search results | 43 |
| 6.3 | Test case-enterprise specific | 44 |
| 6.4 | Test case-name specific | 45 |
| 7.1 | Graphical User Interface- Search engine for user | 47 |
| 7.2 | Graphical User Interface – Inserting search query | 47 |
| 7.3 | Graphical User Interface – Dashboard of search results | 48 |
| 7.4 | Content crawled using nutch crawler and elasticsearch | 50 |
|  | Test Case 1 | 51 |
| 7.5 | Test Case – Input 1 -concept specific search query | 51 |
| 7.6 | Test Case – Output 1 -Dashboard of search results along with twitter trends | 51 |
|  | Test Case 2 | 52 |
| 7.7 | Test Case – Input 1 -name specific search query | 52 |
| 7.8 | Test Case – Output 1 -Dashboard of search results along with twitter trends | 52 |

1. PROJECT OVERVIEW

In today's modern world of technology user require the desired results very quick and efficient. Searching for a particular should be easy so that user can access qucikly and learn the things in lesser time. There are various search engines like google, yahoo,bing which provide to the user the most relevant search results. But these results are mostly ranked on the basis of page rank algorithm which sometimes do not have the relevant content according to the search query. While various marketing strategies are used by websites inorder to rank higher on the results shown by the search engine. There various techniques like black hat,white hat techniques. This techniques takes the user away from the desired search and keeps the user wandering around the irrelevant sites.

Inoder to avoid this,our project tries to create a percipient and cognitive search engine that provides user most relevant content to the desired search query. It eases the process of searching by avoiding the user to go through all the documents in the search results. The search engine directly provides the most relevant paragraphs or text by scanning all the documents in the indexed search results. The key feature of this search engine is that all the meta tags are removed and only the content or the text is scanned on a particular site or a page. Sometimes by increasing the tags or repeating the text inorder to rank the site high in the search results these techniques are used. Mainly the search engine tries to focus on relevant content so that user can easily access it.

2. INTRODUCTION AND MOTIVATION

First thing that we refer to on the internet when we need to get some information on a

topic is search on a search engine which ranks websites based on the views and popularity with

a Page Rank algorithm that makes use of various factors that mainly consider the relativity to

various and the user usage and reference. Previously ranked high rated and valued domains

and keywords tend to be in the top search results.

A website with higher Page Rank tends to appear higher in the results and it doesn’t always

imply efficient and effective results. A student looking for details on a particular topic might be

suggested a result from Yahoo Answers which necessarily answer his doubts and gives an

informative theory about his topic.

But a website on the second page might have more text that is more relative to his search query and might provide him better understanding of the topic. This project revolves around the idea of providing a dashboard for the user where he can get important paragraphs and texts from a pool of relative websites and present it to the user based on his search query. This allows the user to have more relative images and videos and text on the website and thus being more efficient than the conventional search pages.

The project is an attempt to build a smarter search engine. Existing search engines use extensive

crawlers and index these results based on the content, metadata and the sites that link to it.

When a user enters a search query in an existing search engine, he is presented with a number

of relevant links which are ranked by some proprietary algorithms (eg: PageRank) and a few

sentences from that page containing the keyword.

Our project tries to ease the process of searching for a particular topic on the internet. The

approach to search that has been noticed in students researching a particular topic is to open

up numerous tabs and then go through it manually.

Our project will serve as a dashboard for all the results. It will use the index of existing search

engines, but it will present to the user a summary of the content of the top results. It will

summarize the content with emphasis on the keyword/s in the search query. It will also mine and

summarize academic papers that are freely available. All these summaries will have links to their

full pages.

If it is a current hot-topic, it will have presence on social media such as Twitter too. We will include

an option to search Twitter feeds, which will display to the user prominent tweets (based on no.

of RTs/favorites/likes) with a brief sentiment analysis on that feed.

# 1. Theory behind the Project Concept

A search engine should be such that it should provide the most relevant content against the search query entered. It should be percipient and cognitive i.e having good insight of the topic or understanding the entire topic and cognitive is process of learning so that user can easily understands the desired topic.

A search engine operates in the following order:

1. Web crawling
2. Indexing
3. Searching

Web search engines work by storing information about many web pages, which they retrieve from the HTML markup of the pages. These pages are retrieved by a Web crawler (sometimes also known as a spider) — an automated Web crawler which follows every link on the site. The site owner can exclude specific pages by using robots.txt.

The search engine then analyzes the contents of each page to determine how it should be indexed (for example, words can be extracted from the titles, page content, headings, or special fields called meta tags). Data about web pages are stored in an index database for use in later queries. A query from a user can be a single word. The index helps find information relating to the query as quickly as possible. Some search engines, such as Google, store all or part of the source page (referred to as a cache) as well as information about the web pages, whereas others, such as AltaVista, store every word of every page they find.[*citation needed*] This cached page always holds the actual search text since it is the one that was actually indexed, so it can be very useful when the content of the current page has been updated and the search terms are no longer in it.This problem might be considered a mild form of linkrot, and Google's handling of it increases usability by satisfying user expectations that the search terms will be on the returned webpage. This satisfies the principle of least astonishment, since the user normally expects that the search terms will be on the returned pages. Increased search relevance makes these cached pages very useful as they may contain data that may no longer be available elsewhere.

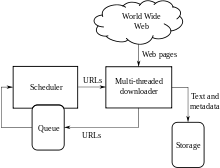


Fig no 1.1 High-level architecture of a standard Web crawler

When a user enters a query into a search engine (typically by using keywords), the engine examines its index and provides a listing of best-matching web pages according to its criteria, usually with a short summary containing the document's title and sometimes parts of the text. The index is built from the information stored with the data and the method by which the information is indexed. From 2007 the Google.com search engine has allowed one to search by date by clicking "Show search tools" in the leftmost column of the initial search results page, and then selecting the desired date range. Most search engines support the use of the boolean operators AND, OR and NOT to further specify the search query. Boolean operators are for literal searches that allow the user to refine and extend the terms of the search. The engine looks for the words or phrases exactly as entered. Some search engines provide an advanced feature called proximity search, which allows users to define the distance between keywords.There is also concept-based searching where the research involves using statistical analysis on pages containing the words or phrases you search for. As well, natural language queries allow the user to type a question in the same form one would ask it to a human. A site like this would be ask.com.

The usefulness of a search engine depends on the relevance of the **result set** it gives back. While there may be millions of web pages that include a particular word or phrase, some pages may be more relevant, popular, or authoritative than others. Most search engines employ methods to rank the results to provide the "best" results first. How a search engine decides which pages are the best matches, and what order the results should be shown in, varies widely from one engine to another.The methods also change over time as Internet usage changes and new techniques evolve. There are two main types of search engine that have evolved: one is a system of predefined and hierarchically ordered keywords that humans have programmed extensively. The other is a system that generates an "inverted index" by analyzing texts it locates. This first form relies much more heavily on the computer itself to do the bulk of the work.

Most Web search engines are commercial ventures supported by advertising revenue and thus some of them allow advertisers to have their listings ranked higher in search results for a fee. Search engines that do not accept money for their search results make money by running search related ads alongside the regular search engine results. The search engines make money every time someone clicks on one of these ads.

In the summer of 1993, no search engine existed for the web, though numerous specialized catalogues were maintained by hand. Oscar Nierstrasz at the University of Geneva wrote a series of Perl scripts that periodically mirrored these pages and rewrote them into a standard format. This formed the basis for W3Catalog, the web's first primitive search engine, released on September 2, 1993.

In June 1993, Matthew Gray, then at MIT, produced what was probably the first web robot, the Perl-based World Wide Web Wanderer, and used it to generate an index called 'Wandex'. The purpose of the Wanderer was to measure the size of the World Wide Web, which it did until late 1995. The web's second search engine Aliweb appeared in November 1993. Aliweb did not use a web robot, but instead depended on being notified by website administrators of the existence at each site of an index file in a particular format.

JumpStation (created in December 1993 by Jonathon Fletcher) used a web robot to find web pages and to build its index, and used a web form as the interface to its query program. It was thus the first WWW resource-discovery tool to combine the three essential features of a web search engine (crawling, indexing, and searching) as described below. Because of the limited resources available on the platform it ran on, its indexing and hence searching were limited to the titles and headings found in the web pages the crawler encountered.

One of the first "all text" crawler-based search engines was WebCrawler, which came out in 1994. Unlike its predecessors, it allowed users to search for any word in any webpage, which has become the standard for all major search engines since. It was also the first one widely known by the public. Also in 1994, Lycos (which started at Carnegie Mellon University) was launched and became a major commercial endeavor.

Soon after, many search engines appeared and vied for popularity. These included Magellan, Excite, Infoseek, Inktomi, Northern Light, and AltaVista. Yahoo! was among the most popular ways for people to find web pages of interest, but its search function operated on its web directory, rather than its full-text copies of web pages. Information seekers could also browse the directory instead of doing a keyword-based search.

In 1996, Netscape was looking to give a single search engine an exclusive deal as the featured search engine on Netscape's web browser. There was so much interest that instead Netscape struck deals with five of the major search engines: for $5 million a year, each search engine would be in rotation on the Netscape search engine page. The five engines were Yahoo!, Magellan, Lycos, Infoseek, and Excite.

Google adopted the idea of selling search terms in 1998, from a small search engine company named goto.com. This move had a significant effect on the SE business, which went from struggling to one of the most profitable businesses in the internet.

Around 2000, Google's search engine rose to prominence. The company achieved better results for many searches with an innovation called PageRank, as was explained in the paper *Anatomy of a Search Engine* written by Sergey Brin and Larry Page, the later founders of Google. This iterative algorithm ranks web pages based on the number and PageRank of other web sites and pages that link there, on the premise that good or desirable pages are linked to more than others. Google also maintained a minimalist interface to its search engine. In contrast, many of its competitors embedded a search engine in a web portal. In fact, Google search engine became so popular that spoof engines emerged such as Mystery Seeker.

By 2000, Yahoo! was providing search services based on Inktomi's search engine. Yahoo! acquired Inktomi in 2002, and Overture (which owned AlltheWeb and AltaVista) in 2003. Yahoo! switched to Google's search engine until 2004, when it launched its own search engine based on the combined technologies of its acquisitions.

Microsoft first launched MSN Search in the fall of 1998 using search results from Inktomi. In early 1999 the site began to display listings from Looksmart, blended with results from Inktomi. For a short time in 1999, MSN Search used results from AltaVista were instead. In 2004, Microsoft began a transition to its own search technology, powered by its own web crawler (called msnbot).

Microsoft's rebranded search engine, Bing, was launched on June 1, 2009. On July 29, 2009, Yahoo! and Microsoft finalized a deal in which Yahoo! Search would be powered by Microsoft Bing technology.

Our project aims to develop a website that will show the related and concise content of

the processed web pages and provide the user a complete webpage with related details and

data.

Objectives:

- Get top search results.

- Get most relevant content from webpages.

- Get relevant tweets and trends from social media.

- Efficient storage of data and processed web pages in the database.

- Use of an efficient and fast algorithm to analyze web pages and process data.

- Display relevant images and videos.

- Display trending tweets related to the user query.

# 2. Problem Definition

The project is an attempt to build a smarter search engine. Existing search engines use extensive crawlers and index these results based on the content, metadata and the sites that link to it.

When a user enters a search query in an existing search engine, he is presented with a number

of relevant links which are ranked by some proprietary algorithms (eg: PageRank) and a few

sentences from that page containing the keyword.

Our project tries to ease the process of searching for a particular topic on the internet. The

approach to search that has been noticed in students researching a particular topic is to open

up numerous tabs and then go through it manually.

Our project will serve as a dashboard for all the results. It will use the index of existing search

engines, but it will present to the user a summary of the content of the top results. It willsummarize the content with emphasis on the keyword/s in the search query. It will also mine andsummarize academic papers that are freely available. All these summaries will have links to their full pages.

If it is a current hot-topic, it will have presence on social media such as Twitter too. We will includean option to search Twitter feeds, which will display to the user prominent tweets (based on no.of RTs/favorites/likes) with a brief sentiment analysis on that feed.

# 3. Need for Project

when we need to get some information on a topic is search on a search engine which ranks websites based on the views and popularity with a Page Rank algorithm that makes use of various factors that mainly consider the relativity to various and the user usage and reference. Previously ranked high rated and valued domains and keywords tend to be in the top search results.

A website with higher Page Rank tends to appear higher in the results and it doesn’t always

imply efficient and effective results. A student looking for details on a particular topic might be

suggested a result from Yahoo Answers which necessarily answer his doubts and gives an

informative theory about his topic. But a website on the second page might have more text that

is more relative to his search query and might provide him better understanding of the topic.

This project revolves around the idea of providing a dashboard for the user where he can get

important paragraphs and texts from a pool of relative websites and present it to the user based

on his search query.

Our project aims at particulary for a enterprise or a industry as searching a a particular content on a website by going through all the inner links becomes a difficult job.So our project directly provide the most relevant content regarding the search query.Thus reducing the job of the user and making the search easy.

# 4. Challenges

-Our project needs to maintain various domains for covering a large data so that searching provides a better results but storing such huge datasets is most challenging part.

-Finding the most relevant content through the crawled websites real time against the search query.

-YouTube has a ridiculously unfair advantage right now. You can quite literally spam a

video (with terrible links that would get any other site penalized) and rank at the top of the

first page of google. Big loophole there. It's funny that google will penalize you for

spamming your own sites but reward you from spamming theirs.

-Google continues to return results for very low quality sites and content farms like Yahoo!

Answers and WikiHow

-Excessive focus on the algorithm means that entire classes of queries, such as medical

queries, are still overrun with exact match domains and poor quality advice.

-Google chooses to display results as "10 blue links," so it can be hard to see the topics

(or clusters) that results are divided up into. If you search for “Big Data” on Google, you

see 3 links related to the Hadoop or Distributed File System, but they're #7, #8 and #10

and you might not even notice them.

This project aims at providing an efficient search engine that tries to overcome the drawbacks

above and improved the search performed by these search engines. The project aims at finding the most relative text from the top predefined number of results and displaying the most relevant and informative text to the user than providing results based on a page ranking algorithm.

3 . ANALYSIS AND DESIGN

# 1. Software Development

Software development is the development of a software product. The term "software development" may be used to refer to the activity of computer programming, which is the process of writing and maintaining the source code, but in a broader sense of the term it includes all that is involved between the conception of the desired software through to the final manifestation of the software, ideally in a planned and structured process. Therefore, software development may include research, new development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

Software can be developed for a variety of purposes, the three most common being to meet specific needs of a specific client/business, to meet a perceived need of some set of potential users, or for personal use (e.g. a scientist may write software to automate a mundane task). Embedded software development, that is, the development of embedded software such as used for controlling consumer products, requires the development process to be integrated with the development of the controlled physical product.

There are several different approaches to software development, much like the various views of political parties toward governing a country. Some take a more structured, engineering-based approach to developing business solutions, whereas others may take a more incremental approach, where software evolves as it is developed piece-by-piece. Most methodologies share some combination of the following stages of software development:

1. Analysing the problem
2. Market research
3. Gathering requirements for the proposed business solution
4. Devising a plan or design for the software-based solution
5. Implementation (coding) of the software
6. Testing the software
7. Deployment
8. Maintenance and bug fixing

There are significant advantages and disadvantages to the various methodologies, and the best approach to solving a problem using software will often depend on the type of problem. If the

problem is well understood and a solution can be effectively planned out ahead of time, the more "waterfall" based approach may work the best. If, on the other hand, the problem is unique (at least to the development team) and the structure of the software solution cannot be easily envisioned, then a more "extreme" incremental approach may work best.

# 2. Flow of Project

# 3.2.1. Preliminary Survey

# Hadoop :

The following paper titled “An ontology-based approach for semantics ranking of the websearch engines results” had been referred to get insight into the semantic algorithm and howthe pages are ranked by popular search engines so that a better algorithm can be implementedbased on the drawbacks. The abstract from the paper is mentioned below

“This work falls in the areas of information retrieval and semantic web, and aims to improve the evaluation of web search tools. Indeed, the huge number of information on the web as well as the growth of new inexperienced users creates new challenges for information retrieval; certainly the current search engines (such as Google, Bing and Yahoo) offer an efficient way to browse the web content. However, this type of tool does not take into account the semantic driven by the query terms and document words. This paper proposes a new semantic based approach for the evaluation of information retrieval systems; the goal is to increase the selectivity of search tools and to improve how these tools are evaluated. The test of the proposed approach for the evaluation of search engines has proved its applicability to real search tools. The results showed that semantic evaluation is a promising way to improve the performance and behavior of search engines as well as the relevance of the results that they return.”

We also referred a paper titled “A New Algorithm for Inferring User Search Goals with

Feedback Sessions” so that we can understand how search engines find relative keywords and similar search results based on the user searched query. As our project involved machine learning, it was a necessary for us to find relative and close keywords to the user search query so that there can be an efficient and more cognitive output for the user. The abstract of the paper is mentioned below:

“For a broad-topic and ambiguous query, different users may have different search goals when

they submit it to a search engine. The inference and analysis of user search goals can be veryuseful in improving search engine relevance and user experience.

In this paper, we propose a novel approach to infer user search goals by analyzing search engine query logs. First, we propose a framework to discover different user search goals for a query by clustering the proposed feedback sessions. Feedback sessions are constructed from user click-through logs and can efficiently reflect the information needs of users. Second, we propose a novel approach to generate pseudo-documents to better represent the feedback sessions for clustering. Finally,we propose a new criterion )“Classified Average Precision (CAP)” to evaluate the performance of inferring user search goals. Experimental results are presented using user click-through logs from a commercial search engine to validate the effectiveness of our proposed methods.”

Another paper that we referred was titled “Entity Search Strategies for Mashup Applications”.

This paper helped us understand better algorithm and design an efficient algorithm to implement in the project so that it can yield better results for the user and thus being an efficient search engine to concise the results rather than simply showcasing the top results with excerpts. Also entity search strategies are involved to assign weights to the processed data and thus allow for proper and efficient ranking of the data that is being processed.

Programmatic data integration approaches such as mashups have become a viable approach to dynamically integrate web data at runtime. Key data sources for mashups include entity search engines and hidden databases that need to be queried via source-specific search interfaces or web forms. Current mashups are typically restricted to simple query approaches such as using keyword search. Such approaches may need a high number ofqueries if many objects have to be found. Furthermore, the effectiveness of the queries may be

limited, i.e they may miss relevant results.

We therefore propose more advanced search strategies that aim at finding a set of entities with high efficiency and high effectiveness. Our strategies use different kinds of queries that are determined by source-specific query generators. Furthermore, the queries are selected based on the characteristics of input entities.

We introduce a flexible model for entity search strategies that includes a ranking of candidate

queries determined by different query generators. We describe different query generators and

outline their use within four entity search strategies.

These strategies apply different query ranking and selection approaches to optimize efficiency and effectiveness. We evaluate our search strategies in detail for two domains: product search and publication search. The comparison with a standard keyword searchshows that the proposed search strategies provide significant improvements in both domains.

**Tools:**

Following are the tools we have decided to implement for the project considering they being open source and having a great community to help us with any issues we face. Hadoop allows

distributed file storage thus allowing to make the project scalable to a far greater extent. Usage of Apache Spark and Mahout which are the languages and programs that are used to run scripts and maintain a Hadoop database. Also page rank algorithm is used by google which we aim to improve and solve the drawbacks of the algorithm implemented by popular search engines.

5Hadoop: Hadoop is a highly scalable analytics platform for processing large volumes of structured and unstructured data. By large scale, we mean multiple petabytes of data spread across hundreds or thousands of physical storage servers or nodes. Hadoop, developed in 2005 and now an open source platform managed under theApache Software Foundation, uses a concept known as MapReduce that is composed of two separate functions.

The Map step inputs data and breaks it down for processing across nodes within a Hadoop instance. These “worker” nodes may in turn break the data down further for processing. In the Reduce step, the processed data is then collected back together and assembled into a format based on the original query being performed.

To cope with truly massive-scale data analysis, Hadoop’s developers implemented a scale-out

architecture, based on many low-cost physical servers with distributed processing of data

queries during the Map operation. Their logic was to enable a Hadoop system capable ofprocessing many parts of a query in parallel to reduce execution times as much as possible. This can be contrasted with legacy-structured database design that looks to scale up within a single server by using faster processors, more memory and fast shared storage.

Looking at the storage layer, the design aim for Hadoop is to execute the distributed processing with the minimum latency possible. This is achieved by executing Map processing on the node that stores the data, a concept known as data locality. As a result, Hadoop implementations can use SATA drives directly connected to the server, thereby keeping the overall cost of the system as low as possible.

To implement the data storage layer, Hadoop uses a feature known as HDFS or the Hadoop Distributed File System. HDFS is not a file system in the traditional sense and isn’t usually directly mounted for a user to view (although there are some tools available to achieve this), which can sometimes make the concept difficult to understand; it’s perhaps better to think of it simply Hadoop data store.

**Google Page Rank**:

PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked 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). Other factors like Author Rank can contribute to the importance of an entity.

A PageRank results from a mathematical algorithm based on the webgraph, created by all World Wide Web pages as nodes and hyperlinks as edges, taking into consideration authority hubs such as cnn.com or usa.gov. The rank value indicates an importance of a particular page. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined re cursively and depends on the number and PageRank metric of all pages that link to it

("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself.

Numerous academic papers concerning PageRank have been published since Page and Brin's original paper. In practice, the PageRank concept may be vulnerable to manipulation. Research has been conducted into identifying falsely influenced PageRank rankings. The goal is to find an effective means of ignoring links from documents with falsely influenced PageRank

# 3.2.2 Feasibility Study

**a. Economic Feasibility :**

Economic analysis most frequently used for evaluating the effectiveness of the system.

More commonly known as cost/benefit analysis the procedure is to determine the benefit and saving that are expected from the system and compare them with costs , decisions is made to design and implement the system.

The project is an attempt to build a smarter search engine. Existing search engines use extensive crawlers and index these results based on the content, metadata and the sites that link to it. Our project tries to ease the process of searching for a particular topic on the internet by providing relevant links. HDFS is designed to be a scalable, fault-tolerant, distributed storage system that works closely with Map Reduce which is open source and freely available. Hence this project makes it efficient and economically feasible to use.

**b. Technical Feasibility :**

Technical feasibility canters on the existing manual system of the management process and to

what extent it can support the system. According to the feasibility analysis procedure, the

technical feasibility of the system is analyzed and the technical requirements such as hardware

requirements are identified.

**c. Behavioral Feasibility :**

An estimate should be made of how strong the smart search engine would prove effective in

making the search based on the user query.

# 3.2.3 Cost Analysis

We are implementing this project of ours by making use of the Apache Hadoop the

distributed processing of large data sets and we are using hard disks for the data storage.

Cost Benefit Analysis is a term that refers both to helping to appraise or assess the case for a project program or policy proposal. Under both definitions, the process involves whether explicitly or implicitly weighing the total expected cost against the total expected benefits of one or more actions inorder to choose the best or the most profitable option. The formal process is mostly referred to as “CBA”(Cost Benefit Analysis) CBA has two purposes:

1. To determine if it is a sound investment/decision(feasibility)

2. To provide a basis for comparing projects. It involves comparing the total expected cost of each option against the total expected benefits, to see whether the benefits outweigh the costs and by how much.

Function Point Calculation:

The estimated FP is derived using the formula:

Function Point Estimation uses functionality as the metric, since functionality cannot be measured directly, it is measured indirectly using other measures. Function Points are derived using empirical relation based on direct measures of software domain and assessment of software complexity, FP based estimation focuses on the information domain values such as inputs, outputs, inquires, files and external interfaces for Class. For the purpose of this estimate, the complexity factor is assumed to be average.

FP = COUNT\_TOTAL \* [ 0.65 + ( 0.01 + ∑ Fi ) ]

FP Based Estimation:

|  |  |
| --- | --- |
| **PARAMETERS** | **WEIGHT** |
| Data Backup and Recovery | 4 |
| Data Communication | 2 |
| Distributed Processing | 1 |
| System Performance | 5 |
| Existing Operating Environment | 4 |
| Online Data Entry | 2 |
| Input Transactions over Multiple Screens | 1 |
| Master File Updated Online | 2 |
| Inputs, Outputs, Files or Enquiries are complex | 3 |
| Internal Processing Complex | 5 |
| Code Designed Reusable | 4 |
| Conversion and installation included in design | 3 |
| Multiple Installation | 2 |
| Application Designed to facilitate Change | 4 |

Fig 3.1 Estimation Table

∑ Fi = 42

Complexity Adjustment Factor [ 0.65 + ( 0.01 \* ∑ Fi ) ] = 1.07

Finally the estimation of FP is derived as:

FP = COUNT\_TOTAL \* [ 0.65 + ( 0.01 + ∑ Fi ) ]

FP = 109\* 1.07

FP = 116.63

# 3.2.4 Process Model

The sequential phases in Waterfall model are: **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

1. **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
2. **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
3. **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
4. **Deployment of system:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
5. **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

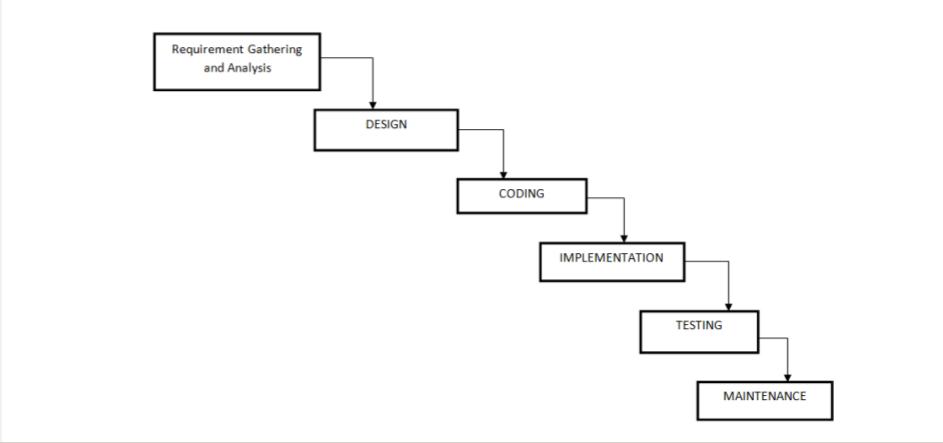
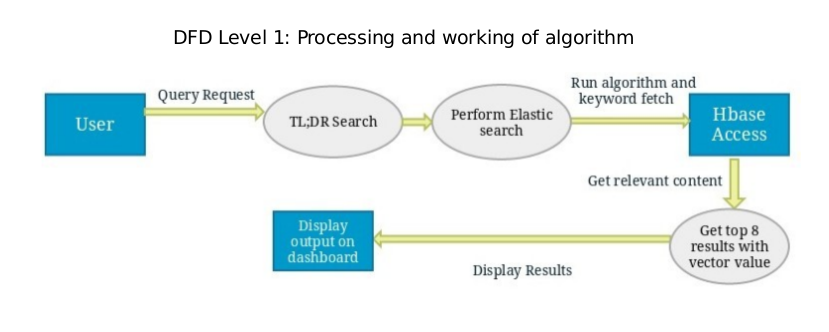


Fig 3.2 Waterfall Model

# 3.2.5 Data Flow Diagrams

# 

Fig 3.3 Data flow diagram(Part I)

Fig 3.4 Data flow diagram (part II)

# 3. UML Diagrams

**I. Usecase diagram:**

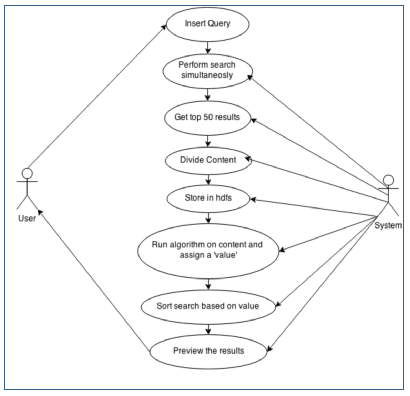


Fig 3.6 Usecase Diagram

**II. Component Diagram:**

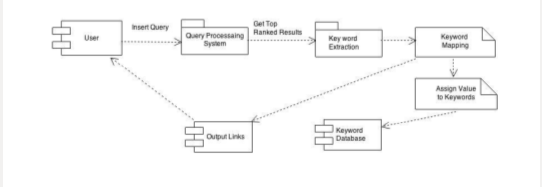


fig 3.7 ComponentDiagram

# 4. Technologies Used

# 4.1 Hardware & Software Requirement

# 

Fig 3.8 Hardware requirements

1. Apache hadoop hbase
2. Apache nutch crawler
3. Beaautiful soup
4. Indexer Elasticsearch
5. Xampp server
6. PHP/jQuey/Python

# 4.2 Introduction to Programming Tools

**I. Apache nutch :**

Apache Nutch is a crawler written in Java. It is continuously updated and its pluggable architecture makes it quite adaptable, which is the reason why we chose Apache Nutch for data extraction. The documentation is not really up to date but the source is good enough to figure out how to customize its behavior.

The whole processing of data extraction in Apache Nutch is done trough plugins. Plugins are a set of extension which corresponds to the extension points defined by Nutch.

The supported extension points by are: OnlineCluster, IndexingFilter, Ontology, Parser,HtmlParserFilter, Protocol, QueryFilter, URLFilter and NutchAnalyzer.

Apache Nutch is a highly extensible and scalable open source web crawler software project. Stemming from Apache Lucene, the project has diversified and now comprises two codebases, namely: Nutch 1.x: A well matured, production ready crawler. 1.x enables fine grained configuration, relying on Apache Hadoop data structures, which are great for batch processing. Nutch 2.x: An emerging alternative taking direct inspiration from 1.x, but which differs in one key area; storage is abstracted away from any specific underlying data store by using Apache Gora for handling object to persistent mappings. This means we can implement an extremely flexibile model/stack for storing everything (fetch time, status, content, parsed text, outlinks, inlinks, etc.) into a number of NoSQL storage solutions. Being pluggable and modular of course has it's benefits, Nutch provides extensible interfaces such as Parse, Index and ScoringFilter's for custom implementations e.g. Apache Tika for parsing. Additonally, pluggable indexing exists for Apache Solr, Elastic Search, etc. Nutch can run on a single machine, but gains a lot of its strength from running in a Hadoop cluster.

**II. Apache Zookeeper:**

All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them ,which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

**III. Elasticsearch :**

Elasticsearch is a tool for querying written words. It can perform some other nifty tasks, but at its core it’s made for wading through text, returning text similar to a given query and/or statistical analyses of a corpus of text.More specifically, elasticsearch is a standalone database server, written in Java, that takes data in and stores it in a sophisticated format optimized for language based searches. Working with it is convenient as its main protocol is implemented with HTTP/JSON. Elasticsearch is also easily scalable, supporting clustering and leader election out of the box.

Whether it’s searching a database of retail products by description, finding similar text in a body of crawled web pages, or searching through posts on a blog, elasticsearch is a fantastic choice. When facing the task of cutting through the semi-structured muck that is natural language, Elasticsearch is an excellent tool.The core of elasticsearch’s intelligent search engine is largely another software project: Lucene. It is perhaps easiest to understand elasticsearch as a piece of infrastructure built around Lucene’s Java libraries. Everything in elasticsearch that pertains to the actual algorithms for matching text and storing optimized indexes of searchable terms is implemented by Lucene. Elasticsearch itself provides a more useable and concise API, scalability, and operational tools on top of Lucene’s search implementation.

**IV. Google Freebase API :**

The use of freebase API is to obtain the related keyowrds regarding the search query entered by the user .This will try to provide the complete knowledge about the search query entered.

The Freebase API is a collection of HTTP APIs that provide read and write access to the data stored in Freebase. The different APIs support different use cases and allow access to the same Freebase data in different ways.

The MQL Read and MQL Write APIs provides access to the Freebase database using the Metaweb query language (MQL).

Some examples of how you would use the MQL Read API include:

Get a collection of entities that share some common attributes or relations.

Get a specific set of facts about an entity.

Count the number of entities in Freebase that match certain criteria.

Query the individual links that make up the graph.

Search Input :

Input to the *search* service takes the form of HTML form-encoded parameters appended to the URL. There are four categories of parameters:

<itemizedlist>

parameters that specify the text to be matched;

parameters that narrow the field of search by domain or type;

parameters that specify the number or offset of the desired results; and

parameters that affect the format of the returned results.

4. PROJECT TIME & TASK DISTRIBUTION

# 1. Timeline Chart

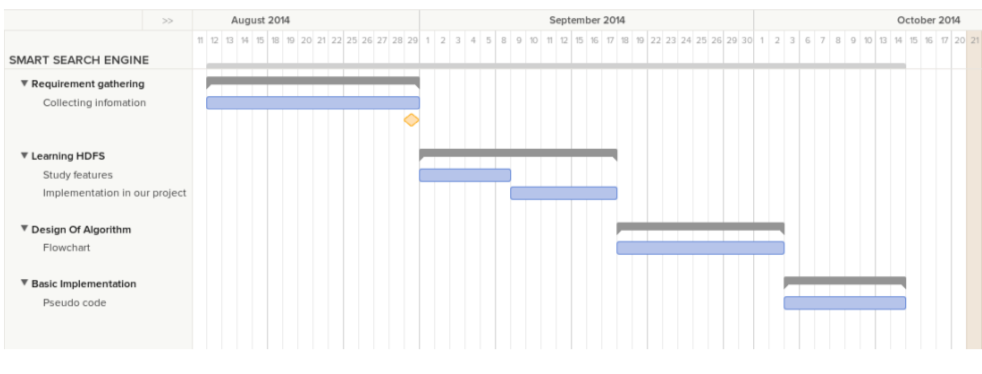


Fig 4.1 Timeline Chart (For Semester 7)

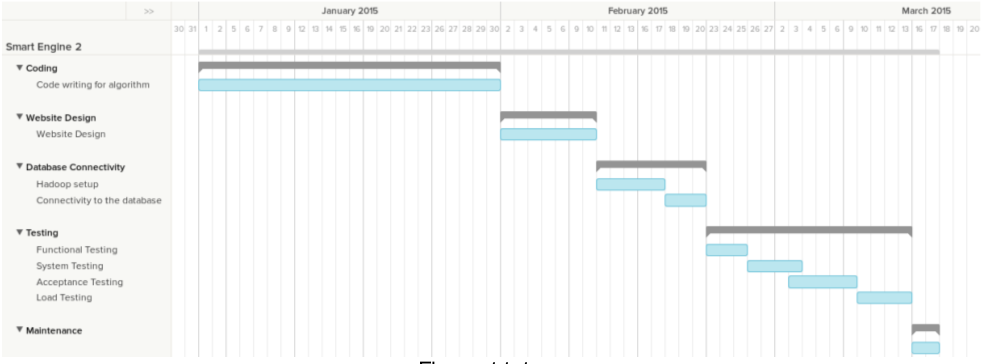


Fig 4.2 Timeline Chart (For Semester 8)

5. IMPLEMENTATION

The complete implementation has been divided into 2 Sections. Section I discusses crawling a particular domain. This section mainly discusses the websites or urls that being crawled. Section II discusses the searching through the documents being crawled. It uses a algorithm inorder to provide the relevant paragraphs.

**SECTION-1**

1. **Intializing Hbase**

The very first step is to intialize the hadoop hbase.Hbase setup is required inorder to store the huge datasets.These datasets are the ones which are formed huge documents and the links within the sites that are crawled.We can maintain one or more domains,so hbase storage is required for these large data.The configuration file need to be changed inorder to create hadoop hbase.

------> Setting up HBase

-> edit $HBASE\_ROOT/conf/hbase-site.xml and add

<configuration>

<property>

<name>hbase.rootdir</name>

<value>home/madhura/Proj/Storage/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>false</value>

</property>

</configuration>

-> edit $HBASE\_ROOT/conf/hbase-env.sh and enable JAVA\_HOME and set it to the proper path:

-# export JAVA\_HOME=/usr/java/jdk1.6.0/

+export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64

> Start HBase using command below:

$HBASE\_ROOT/bin/start-hbase.sh

Hbase running can be verified by the command : [http://localhost:60010](http://localhost:60010/)

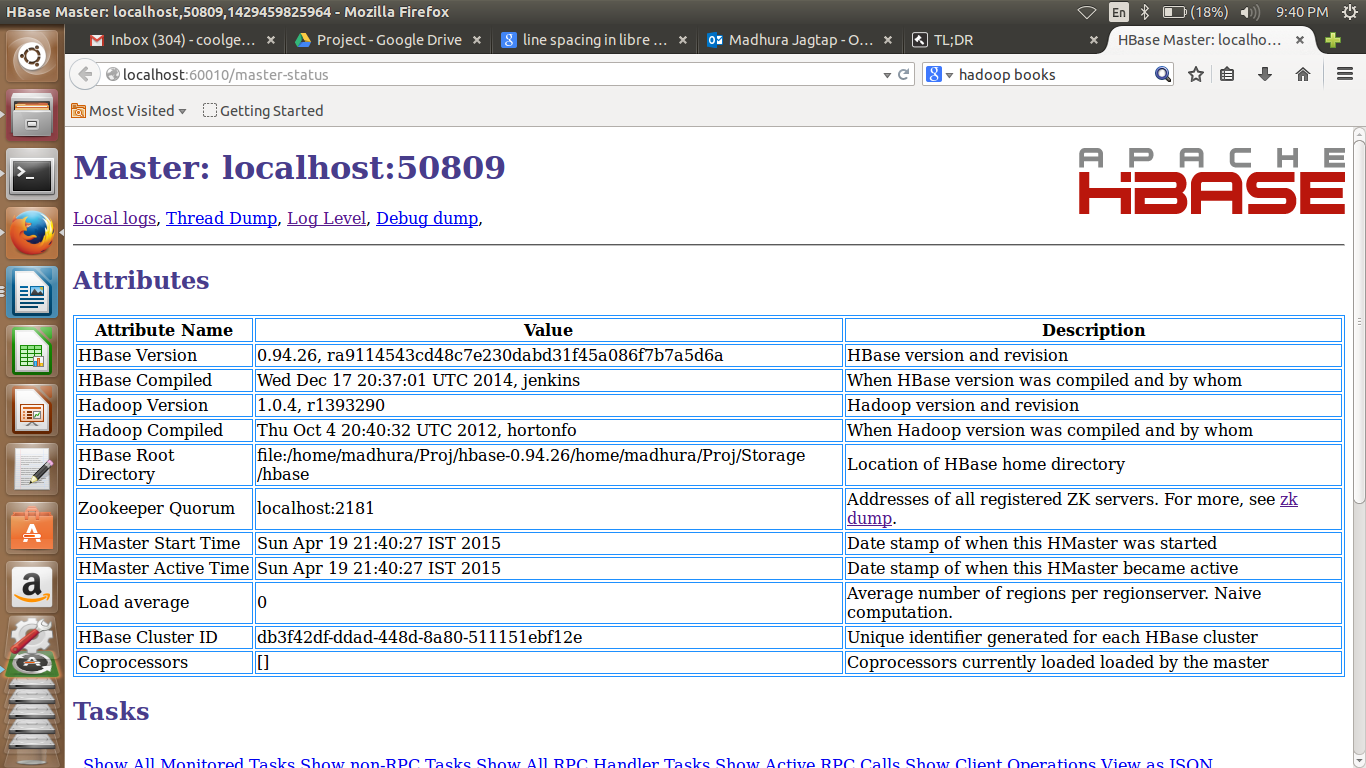


Fig 5.1 hadoop hbase cluster

**II. Initialising nutch crawler**

Crawler made by apache is used inorder to crawl the sites by injecting the various urls.The ulrls crawled is injected to the hbase.

-------> Setting up Nutch

-> enable the HBase dependency in $NUTCH\_ROOT/ivy/ivy.xml by uncommenting the line

<dependency org="org.apache.gora" name="gora-hbase" rev="0.5" conf="\*->default" />

configure the HBase adapter by editing the $NUTCH\_ROOT/conf/gora.properties:

-#gora.datastore.default=org.apache.gora.mock.store.MockDataStore

+gora.datastore.default=org.apache.gora.hbase.store.HBaseStore

-> build Nutch

$ cd $NUTCH\_ROOT

$ ant clean

$ ant runtime

This can take a while(about 20 minutes) and creates $NUTCH\_ROOT/runtime/local.

-> configure Nutch by editing $NUTCH\_ROOT/runtime/local/conf/nutch-site.xml:

<configuration>

<property>

<name>http.agent.name</name>

<value>mycrawlername</value> <!-- this can be changed to something more sane if you like -->

</property>

<property>

<name>http.robots.agents</name>

<value>mycrawlername</value> <!-- this is the robot name we're looking for in robots.txt files -->

</property>

<property>

<name>storage.data.store.class</name>

<value>org.apache.gora.hbase.store.HBaseStore</value>

</property>

<property>

<name>plugin.includes</name>

<!-- do \*\*NOT\*\* enable the parse-html plugin, if you want proper HTML parsing. Use something like parse-tika! -->

<value>protocol-httpclient|urlfilter-regex|parse-(text|tika|js)|index-(basic|anchor)|query-(basic|site|url)|response-(json|xml)|summary-basic|scoring-opic|urlnormalizer-(pass|regex|basic)|indexer-elastic</value>

</property>

<property>

<name>db.ignore.external.links</name>

<value>true</value> <!-- do not leave the seeded domains (optional) -->

</property>

<property>

<name>elastic.host</name>

<value>localhost</value> <!-- where is ElasticSearch listening -->

</property>

</configuration>

-> configure HBase integration by editing $NUTCH\_ROOT/runtime/local/conf/hbase-site.xml:

<configuration>

<property>

<name>hbase.rootdir</name>

<value>home/madhura/Proj/Storage/hbase</value> <!-- same path as you've given for HBase above -->

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>false</value>

</property>

</configuration>

Everything is now setup to crawl websites.

-------> Adding new Domains to crawl with Nutch

-> create an empty directory. Add a textfile containing a list of seed URLs.

$ mkdir seed

$ echo "http://www.vit.edu.in" >> seed/urls.txt

$ echo "https://www.another.com" >> seed/urls.txt

$ echo "https://www.example.com" >> seed/urls.txt

-> inject them into Nutch by giving a file URL (!)

$ $NUTCH\_ROOT/runtime/local/bin/nutch inject file:///path/to/seed/

**III. Crawling websites :**

----> Actual Crawling Procedure

-> Generate a new set of URLs to fetch. This is is based on both the injected URLs as well as outdated URLs in the Nutch crawl db.

$ $NUTCH\_ROOT/runtime/local/bin/nutch generate -topN 9999

The above command will create job batches for 9999 URLs.

-> Fetch the URLs. We are not clustering, so we can simply fetch all batches:

$ $NUTCH\_ROOT/runtime/local/bin/nutch fetch -all

-> Now we parse all fetched pages:

$ $NUTCH\_ROOT/runtime/local/bin/nutch parse -all

-> Last step: Update Nutch's internal database:

$ $NUTCH\_ROOT/runtime/local/bin/nutch updatedb -all

On the first run, this will only crawl the injected URLs. The procedure above is supposed to be repeated regulary to keep the index up to date.

SECTION II

**IV. Setting up elasticsearch and integrating algorithm :**

The algorithm works as follows the keyword is searched through the entire crawled pages, as if now we have cached 500 pages for a enterprise search. Initially the frequency of the keyword in the search query is through the document and assign weights accordingly.Thus after acquiring the weights the highest weighted paragraphs will be displayed in the search results .Thus vectors will be calcuated and most relevant paragraphs will be shown.

All the content crawled is stored in json format thus making it compatible to all other programming languages to work with.

**I. Code:**

// Initiate curl

$ch = curl\_init();

// Disable SSL verification

curl\_setopt($ch, CURLOPT\_SSL\_VERIFYPEER, false);

// Will return the response, if false it print the response

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, true);

// Set the url

curl\_setopt($ch, CURLOPT\_URL,$url);

// Execute

$result=curl\_exec($ch);

// Closing

curl\_close($ch);

$testing123 = json\_decode($result, true);

//echo "Meh";

$variable1 = $testing123["hits"]["hits"][0]["\_source"]["content"];

$sizeoftext1 = strlen($variable1);

if($sizeoftext1 >= 100)

{

$start1 = rand(500, $sizeoftext1-2000);

// $substring1 = "...".substr( $variable1 , $start1, 350 )."...";

//$pos1 = stripos($variable1, $searchquery);

$positionarray1 = strpos\_all($variable1, $searchquery);

if($positionarray1[3]!=NULL)

{

$pos1=$positionarray1[3]-150;

}

else if($positionarray1[2]!=NULL)

{

$pos1=$positionarray1[2]-150;

}

else if($positionarray1[1]!=NULL)

{

$pos1=$positionarray1[1]-150;

}

else

{

$pos1=$positionarray1[0]-150;

}

$substring1 = "...".substr( $variable1 , $pos1, 350 )."...";

$url= $testing123["hits"]["hits"][0]["\_source"]["url"];

$htmlurl1 = "<a href='".$url."'>Visit Site!</a>";

$title1 = $testing123["hits"]["hits"][0]["\_source"]["title"];

if($\_GET['dev']==1)

{

$vecvalue1 = "<br>Search Vector Value: ".$testing123["hits"]["hits"][0]["\_score"];

}

else

{

$vecvalue1 = "";

}

}

else

{

$substring1 = "No Results in Cached Pages";

$title1 = "No Results";

$htmlurl1 = "";

$vecvalue1 = "";

}

**II. Elasticsearch code :**

import string

import elasticsearch

def elastic\_words(elasticsearch):

var1 = elasticsearch->Parse1

var2 = elasticsearch->Parse2

var3 = elasticsearch->Parse3

var4 = elasticsearch->Parse4

var5 = elasticsearch->Parse5

searcharray = [var1, var2, var3, var4, var5];

def find\_max\_letter\_count(word):

alphabet = string.ascii\_lowercase

dictionary = {}

for letters in alphabet:

dictionary[letters] = 0

for letters in word:

dictionary[letters] += 1

dictionary = sorted(dictionary.items(),

reverse=True,

key=lambda x: x[1])

for position in range(0, 26):

print dictionary[position]

if position != len(dictionary) - 1:

if dictionary[position + 1][1] < dictionary[position][1]:

break

finalarray = find\_max\_letter\_count(searcharray)

return finalarray

**IV. Putting Documents into ElasticSearch**

-> Use the following command

$ $NUTCH\_ROOT/runtime/local/bin/nutch index -all

Inorder to view the content of the site being crawled we can start elasticsearch and follow the commands given below:

i . curl '[http://localhost:9200](http://localhost:9200/)'

ii. localhost:9200/\_search?=xyz&pretty=true

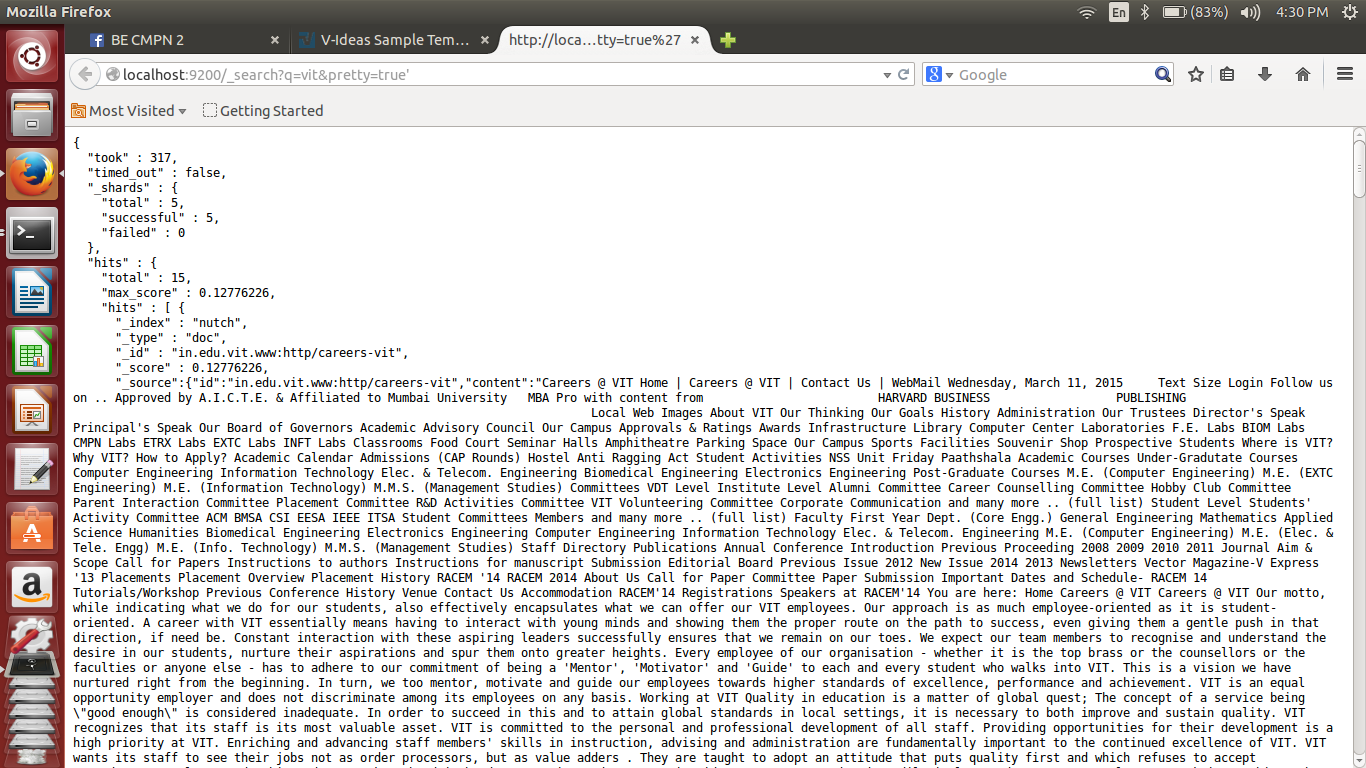


fig 5.2 crawled content without any metatags

6. TEST CASES

The following are the test cases for the browser based user interface and some expected output :

# 6.1 Graphical User Interface

Test Case #1

|  |  |
| --- | --- |
| Purpose: | To ensure that the search engine is running properly on the local machine |

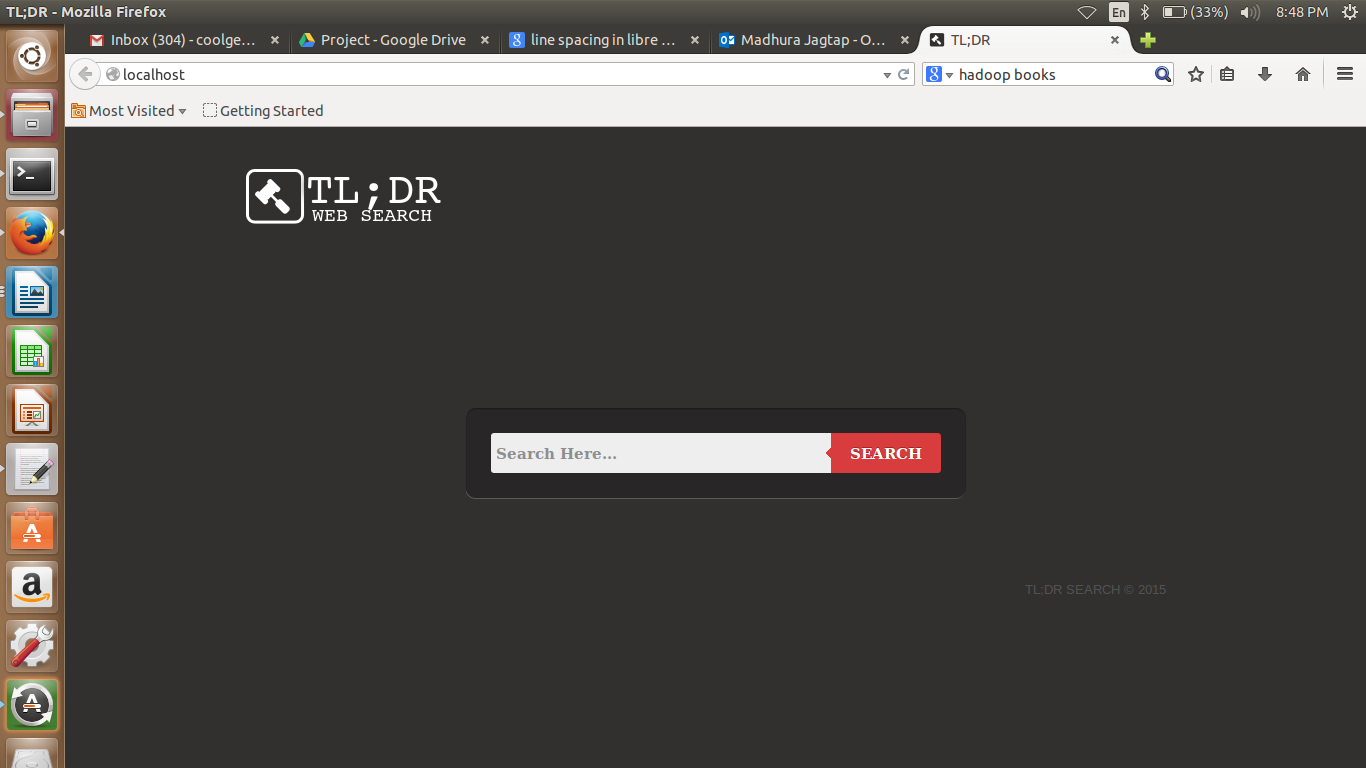


fig 6.1 GUI of the search engine

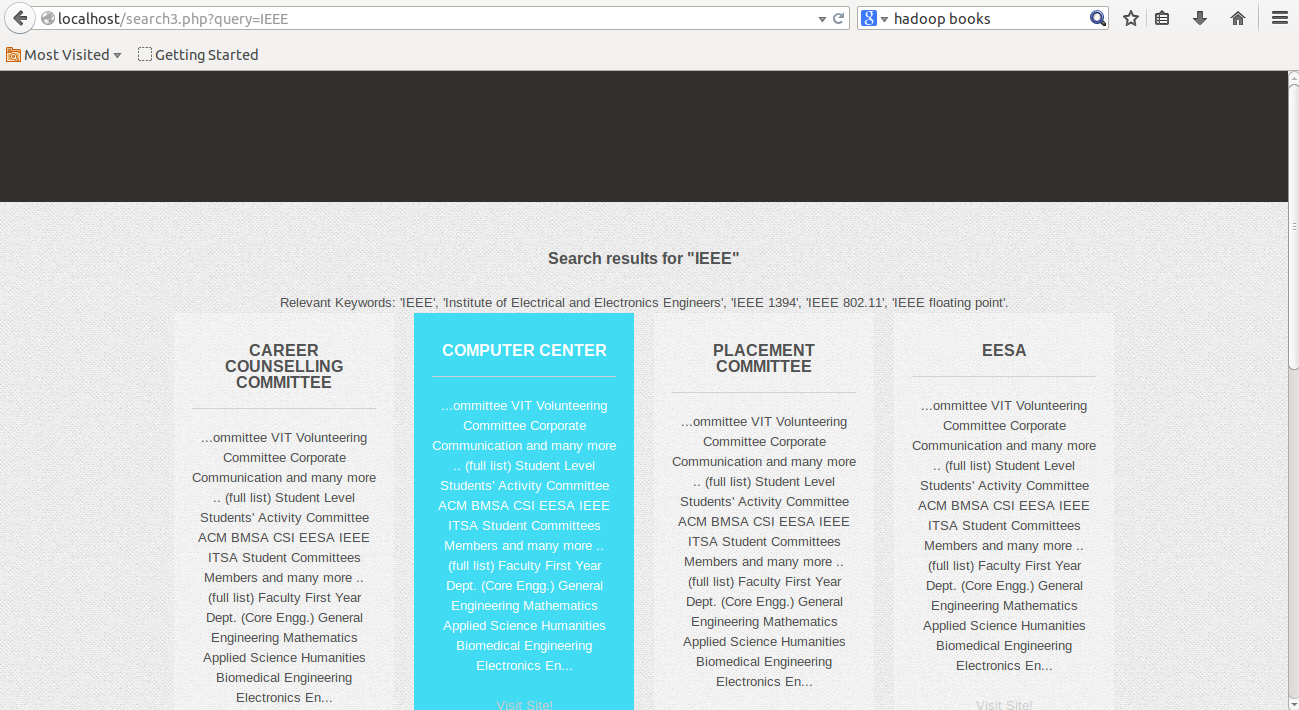


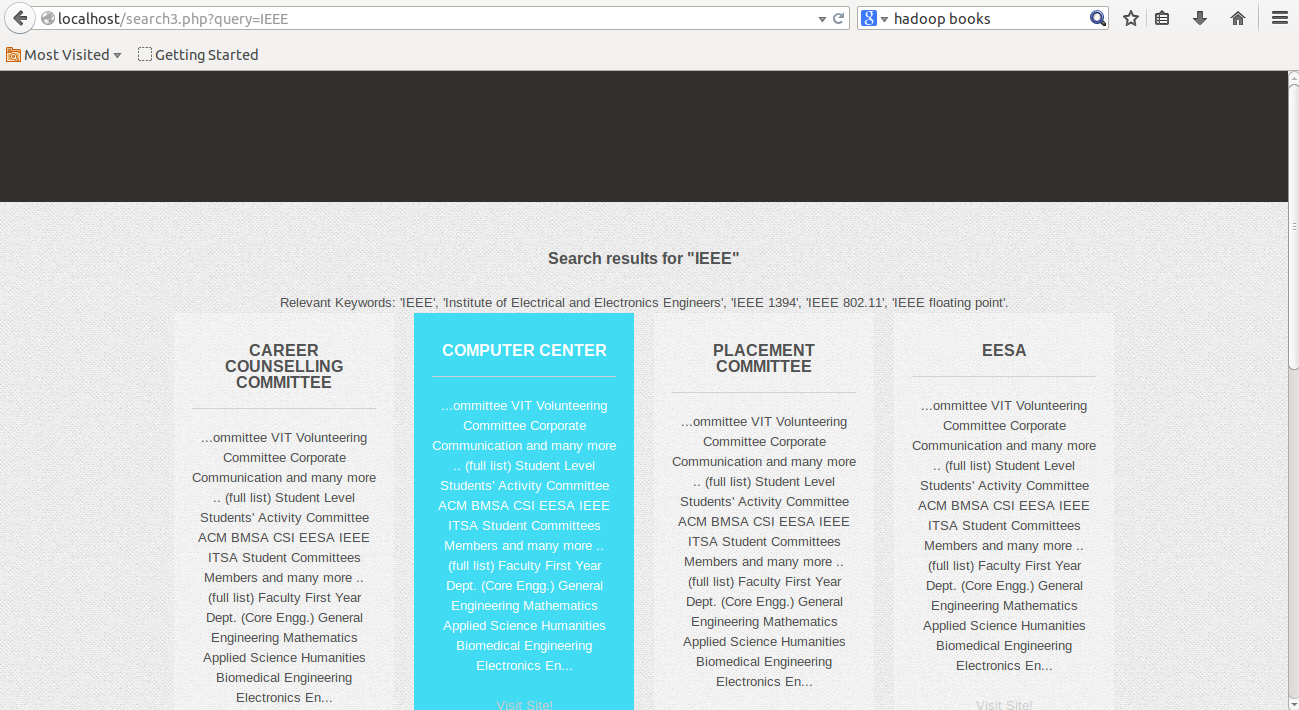
Fig 6.2 GUI of the search results

# 6.2 Test Cases

Test Case #1

|  |  |
| --- | --- |
| 1. Purpose: | 1. To ensure that a relevant content is shown depending on search query for a enterprise |

|  |  |  |
| --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** |
| Inserting a search query regarding a topic in a enterprise | Showing the most relevant content having the search query keywords. | Relevant content in paragraph format is shown for the particular search query. |

Fig 6.3 Test Case –enterpise specific

Test Case #2

|  |  |
| --- | --- |
| PuPurpose: | To ensure that a search which is name specific should show the entire content regarding that particular person. |

|  |  |  |
| --- | --- | --- |
| **Input** | **Expected Output** | **Actual Output** |
| Insert a search query which is name specific | The entire content regarding that person should be visible on the site. Along with his twitter activities should be seen and on the social media sites. | The complete information regarding the person is seen on the site along with the twitter trends or activity. |

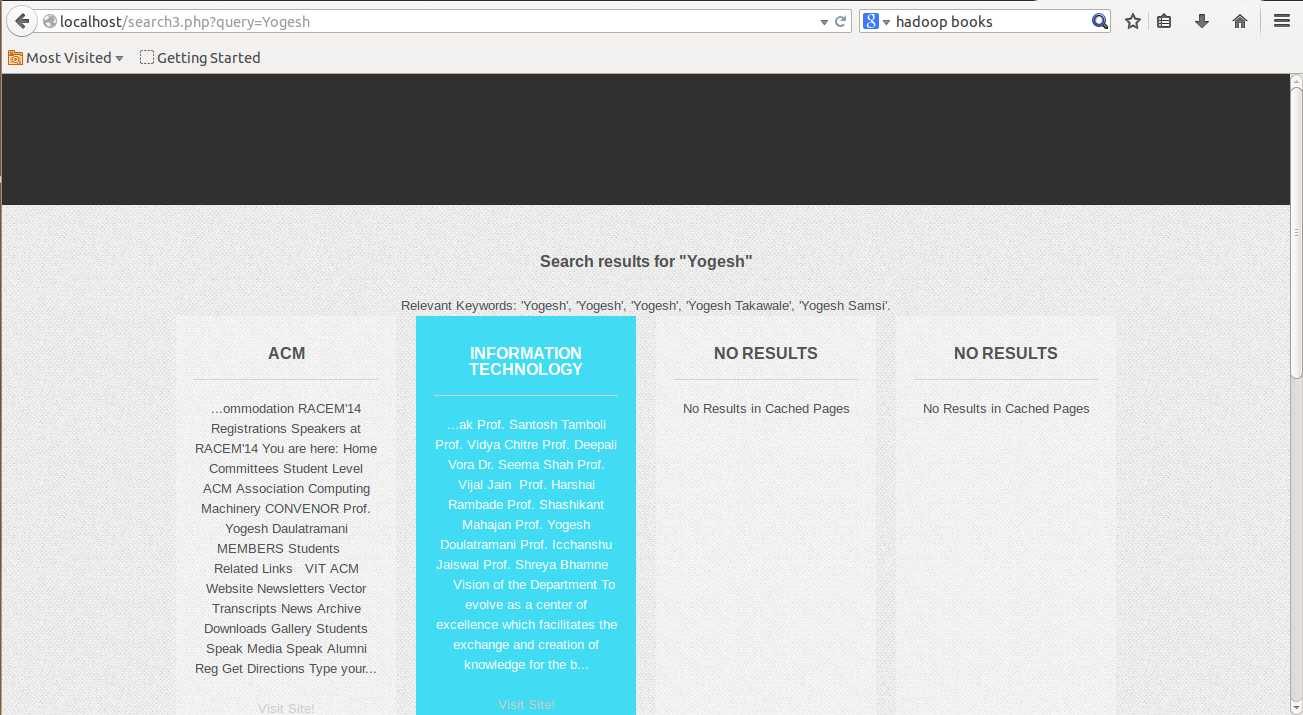


Fig 6.4 Test Case – Name specific

7. CONCLUSION AND FUTURE SCOPE

This project tries to serve a better search engine which would be helpful for a enterprise or organization.

Youtube has a ridiculously unfair advantage right now. You can quite literally spam a video (with terrible links that would get any other site penalized) and rank at the top of the first page of google. Big loophole there. It's funny that google will penalize you for spamming your sites but reward you from spamming theirs. Google continues to return results for very low quality sites and content farms like Yahoo! Answers and WikiHow.

Excessive focus on the algorithm means that entire classes of queries, such as medical queries, are still overrun with exact match domains and poor quality advice.

Google chooses to display results as "10 blue links," so it can be hard to see the topics (or clusters) that results are divided up into. If you search for “Big Data” on Google, you see 3 links related to the Hadoop or Distributed File System, but they're #7, #8 and #10 and you might not even notice them.

This project aims at providing an efficient search engine that tries to overcome the drawbacks above and improved the search performed by these search engines. The project aims at finding the most relative text from the top predefined number of results and displaying the most relevant and informative text to the user than providing results based on a page ranking algorithm.

* APPENDIX A: USER MANUAL
* Start the application through a local server, like XAMPP on the local machine.

* The search engine will be shown inorder to search a particular query.

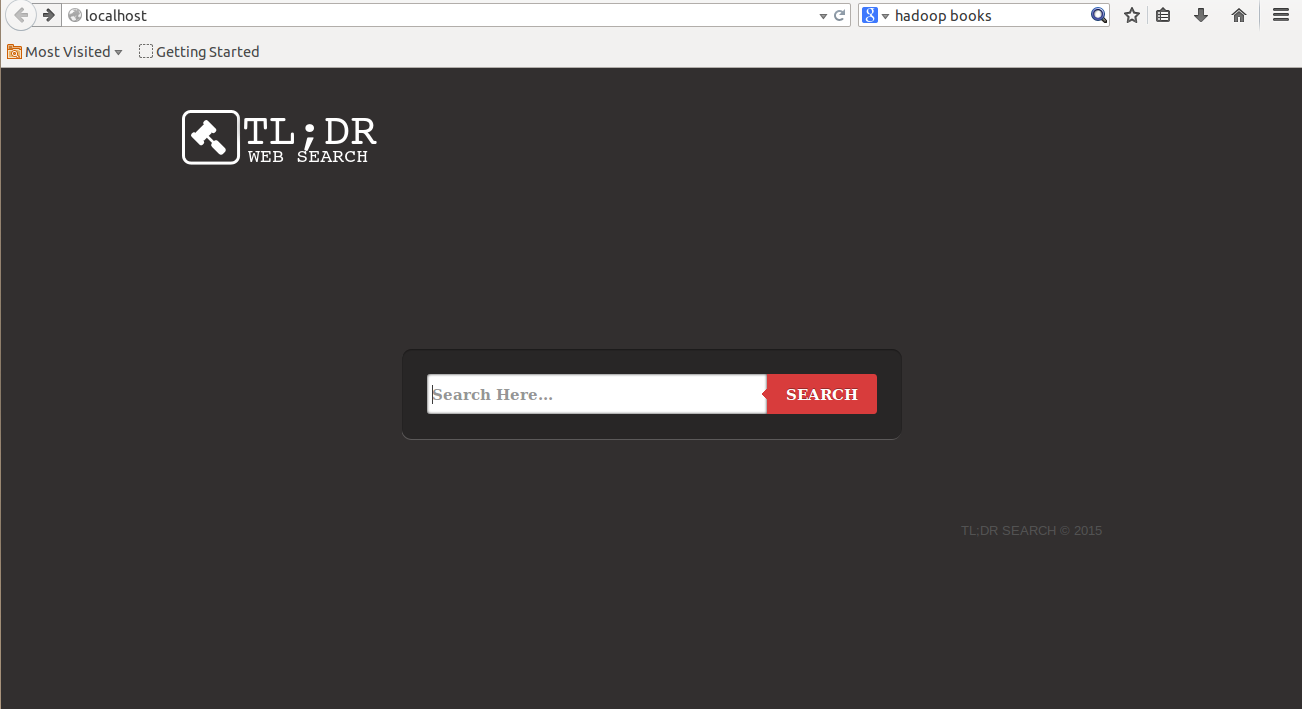


Fig 7.1 Search engine for user

* A search bar is provided for the user to enter the search query.It is just like a simple search engine.It is user-friendly.

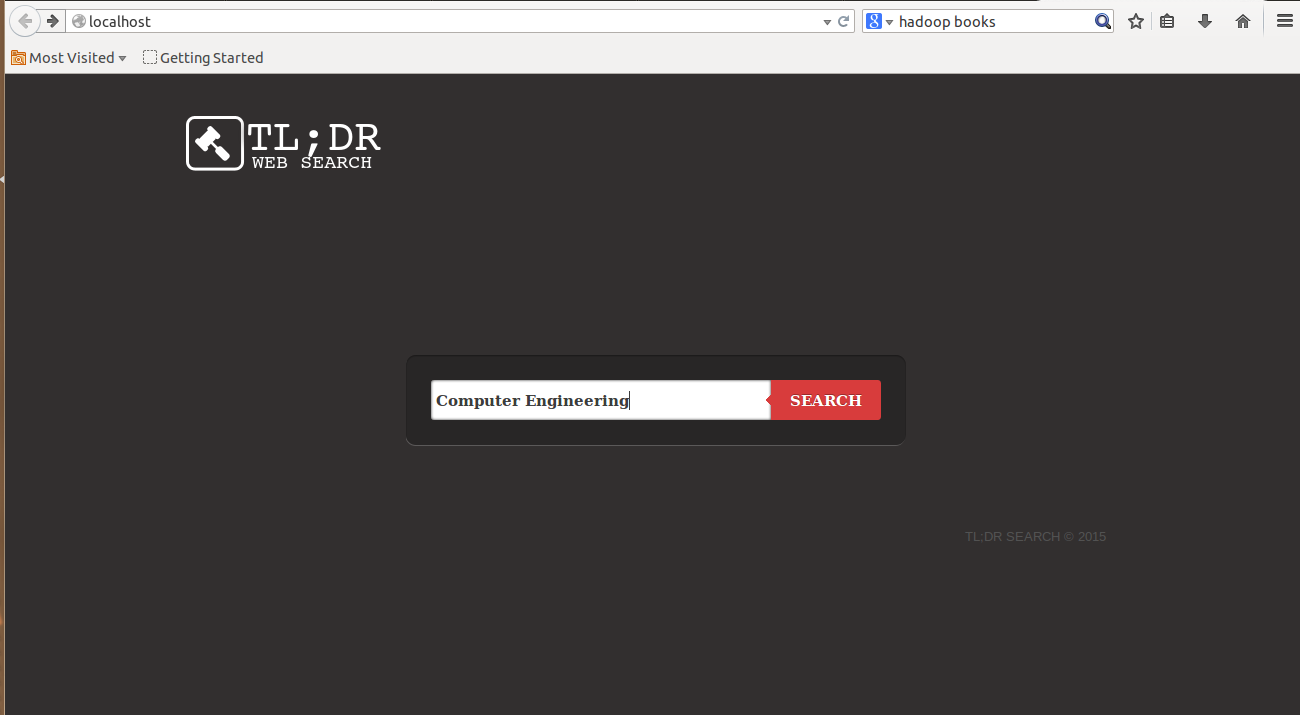


Fig 7.2 Inserting a search query

* After the search query is entered,the search results has a particular pattern like it shows the most relevant paragraphs on the top and also it shows twitter trends or current news regarding the keywords.
* User can through the entire paragraph or document by the hyperlinks provided below the each paragraph.

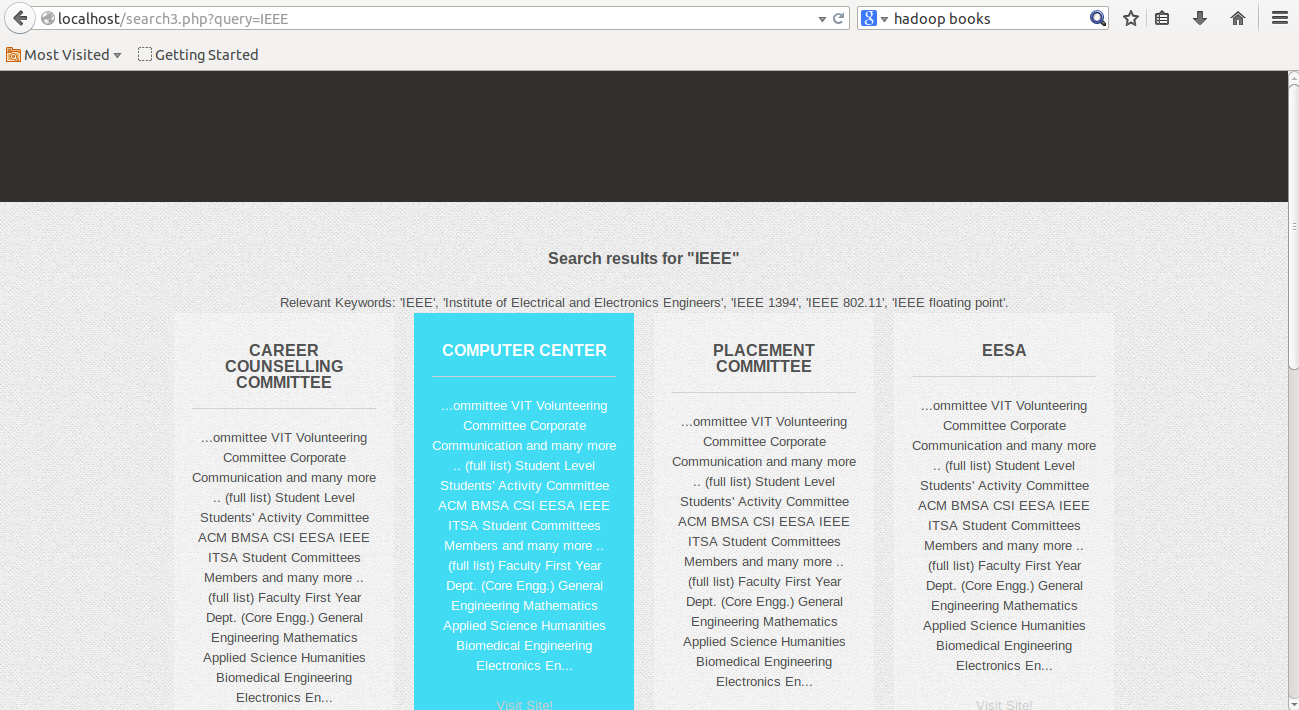


Fig 7.3 Dashboard of search results

* APPENDIX B: CLASSES AND EXTERNAL LIBRARIES,TOOLS

**I. Apache Ant**

**Apache Ant** is a software tool for automating software build processes. It originally came from the Apache Tomcat project in early 2000. It was a replacement for the unix make build tool, and was created due to a number of problems with the unix make.It is similar to Make but is implemented using the Java language, requires the Java platform, and is best suited to building Java projects.

The most immediately noticeable difference between Ant and Make is that Ant uses XML to describe the build process and its dependencies, whereas Make uses Makefile format. By default the XML file is named build.xml.

Ant is an Apache project. It is open source software, and is released under the Apache License.

**II. Elastic search:**

Elasticsearch is a tool for querying written words. It can perform some other nifty tasks, but at its core it’s made for wading through text, returning text similar to a given query and/or statistical analyses of a corpus of text.

More specifically, elasticsearch is a standalone database server, written in Java, that takes data in and stores it in a sophisticated format optimized for language based searches. Working with it is convenient as its main protocol is implemented with HTTP/JSON. Elasticsearch is also easily scalable, supporting clustering and leader election out of the box.

Whether it’s searching a database of retail products by description, finding similar text in a body of crawled web pages, or searching through posts on a blog, elasticsearch is a fantastic choice. When facing the task of cutting through the semi-structured muck that is natural language, Elasticsearch is an excellent tool.

The core of elasticsearch’s intelligent search engine is largely another software project: Lucene. It is perhaps easiest to understand elasticsearch as a piece of infrastructure built around Lucene’s Java libraries. Everything in elasticsearch that pertains to the actual algorithms for matching text and storing optimized indexes of searchable terms is implemented by Lucene. Elasticsearch itself provides a more useable and concise API, scalability, and operational tools on top of Lucene’s search implementation.

**III. Google Freebase API**

The use of freebase API is to obtain the related keyowrds regarding the search query entered by the user .This will try to provide the complete knowledge about the search query entered.

The Freebase API is a collection of HTTP APIs that provide read and write access to the data stored in Freebase. The different APIs support different use cases and allow access to the same Freebase data in different ways.

The MQL Read and MQL Write APIs provides access to the Freebase database using the Metaweb query language (MQL).

Some examples of how you would use the MQL Read API include:

1. Get a collection of entities that share some common attributes or relations.
2. Get a specific set of facts about an entity.
3. Count the number of entities in Freebase that match certain criteria.
4. Query the individual links that make up the graph

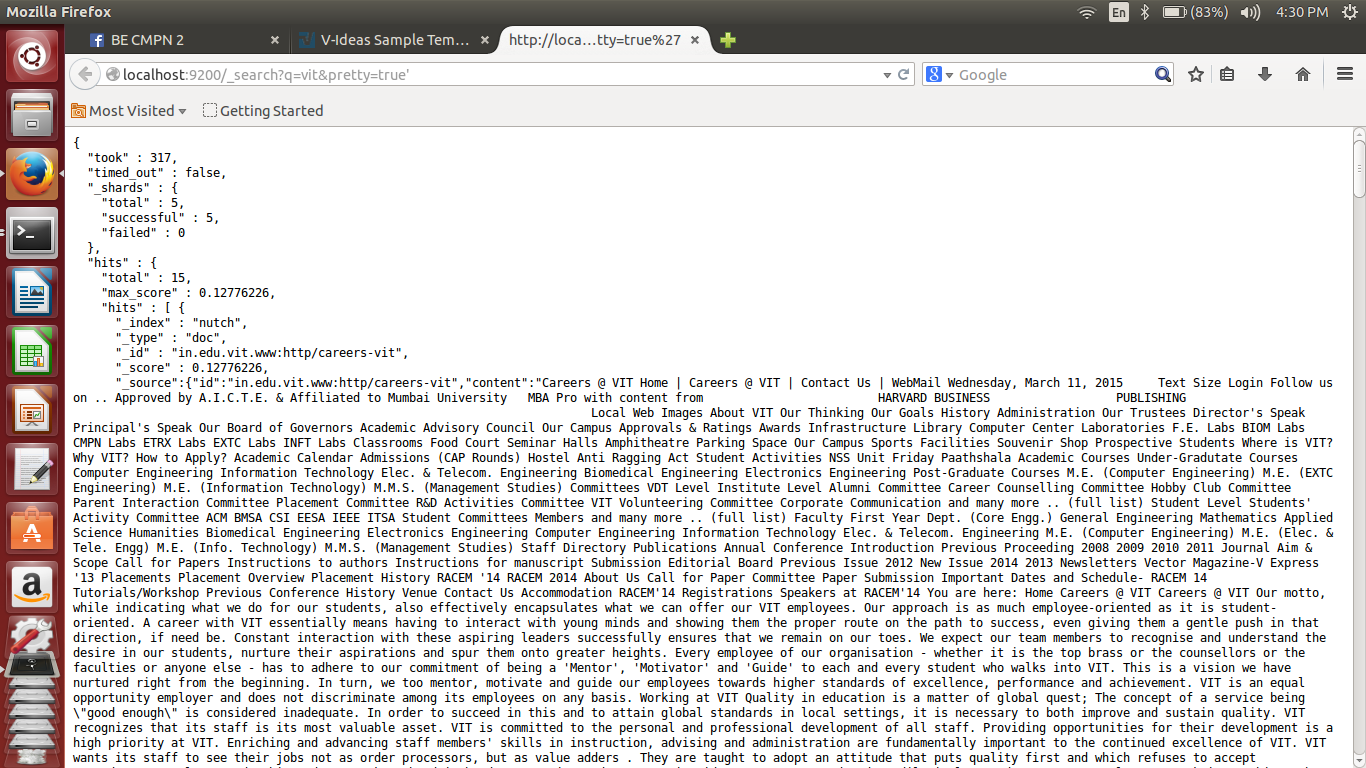


Fig 7.4 Content crawled using nutch crawler and elasticsearch

* APPENDIX C: INPUT AND OUTPUT FOR TEST CASES
* Input 1 :

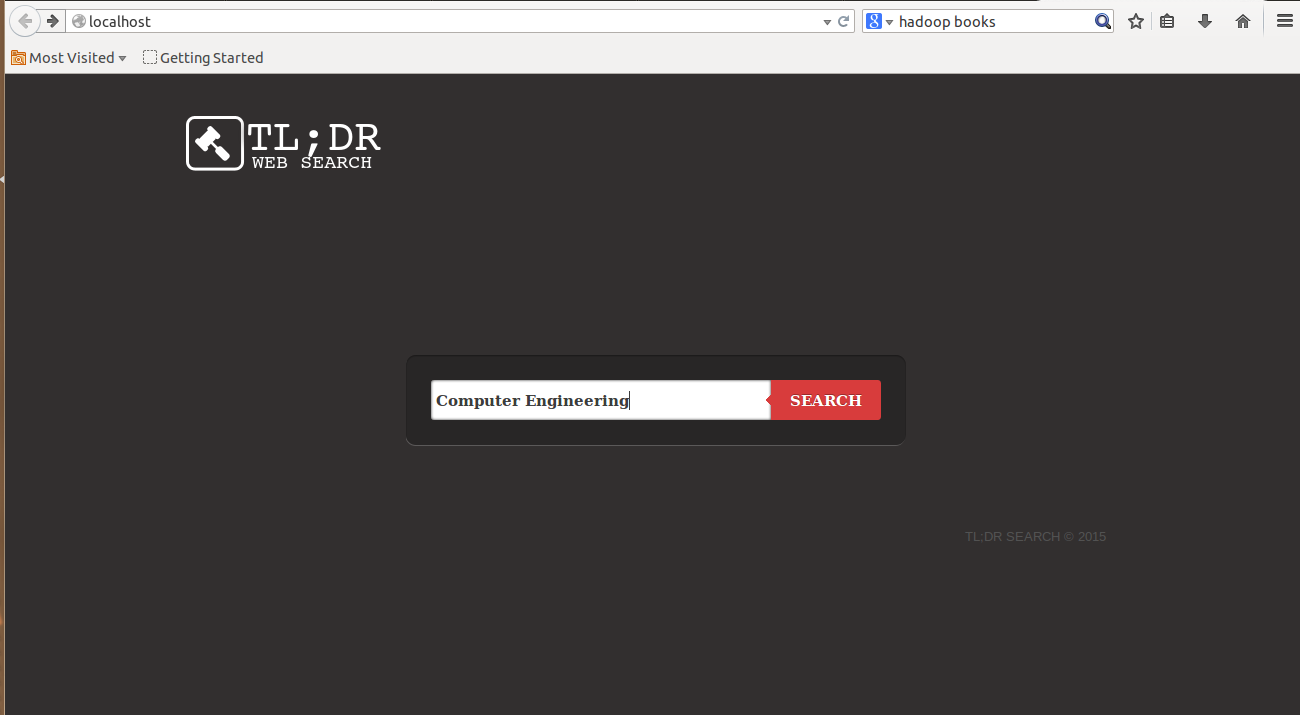


Fig 7.5 Input-1

* Output 1:

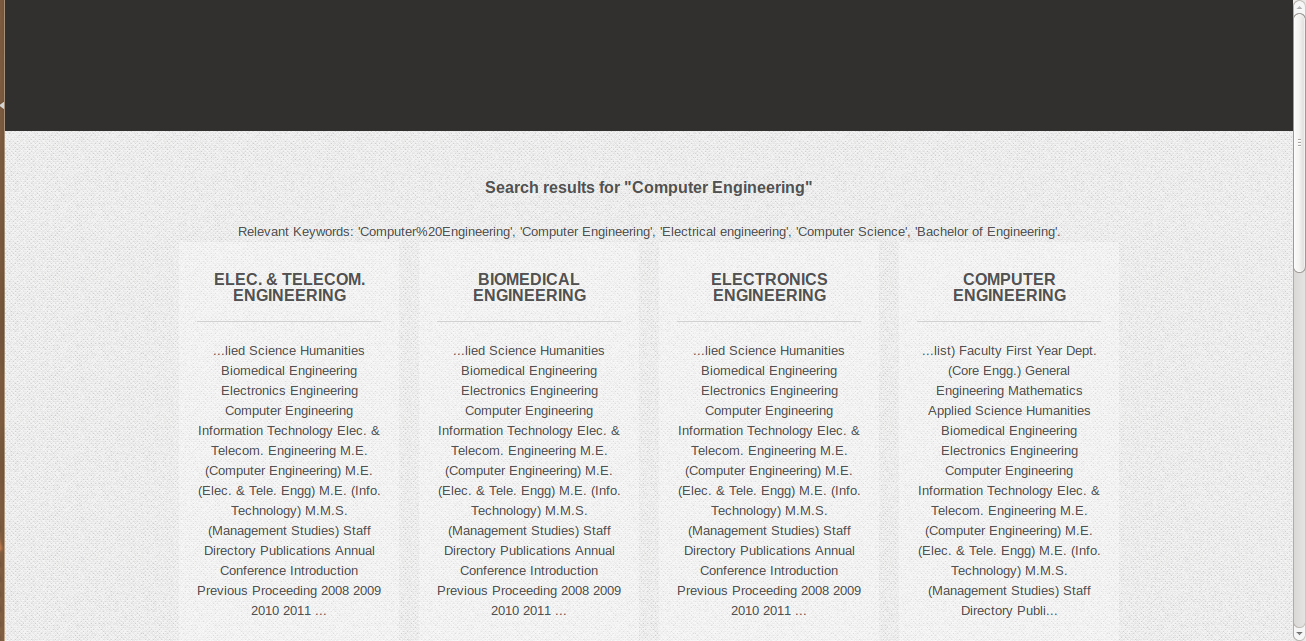


Fig 7.6 Output-1

* Input 2:

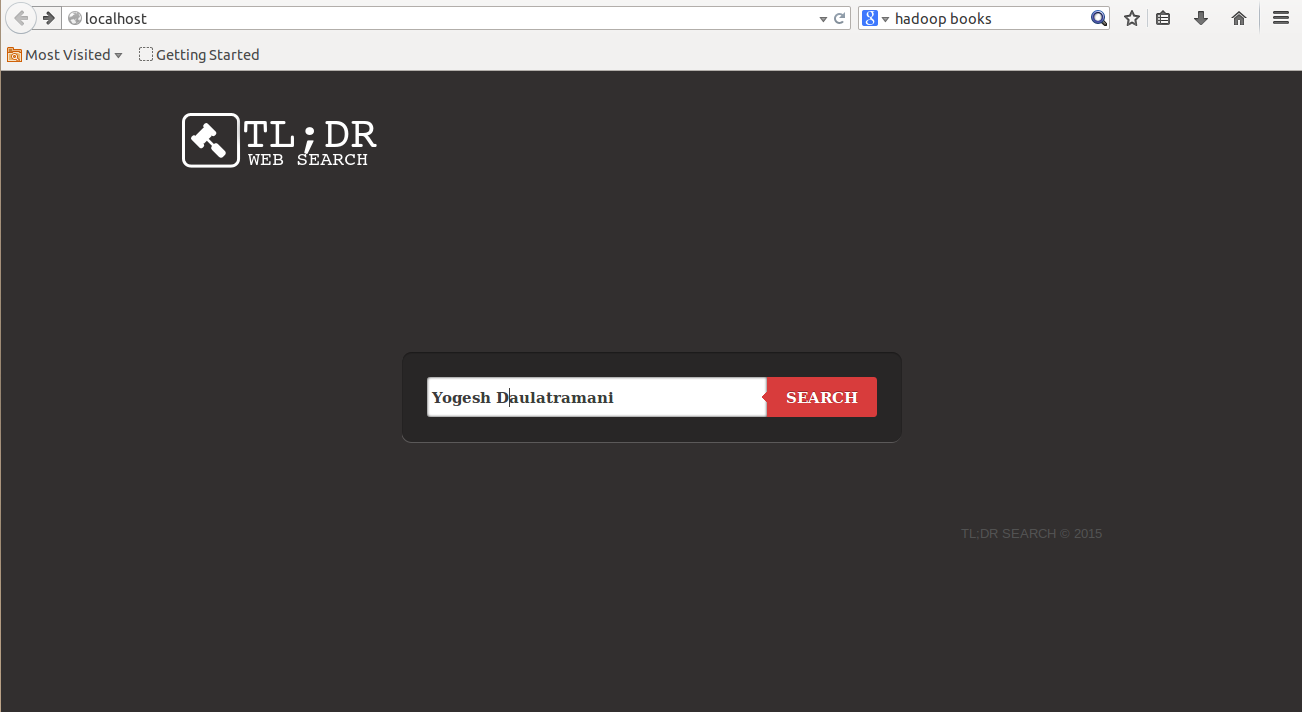


Fig 7.7 Input-2

* Output 2:

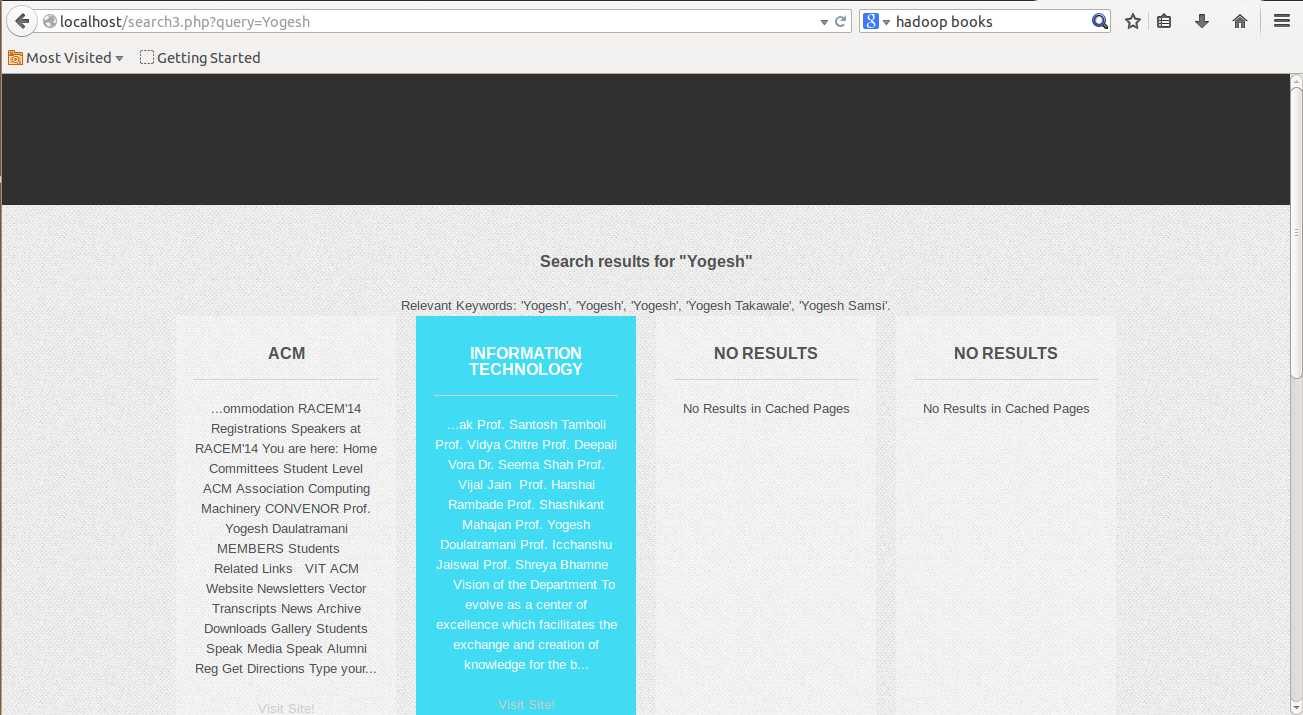


Fig 7.8 Output-2

* BIBLIOGRAPHY

[1] Abiteboul, Serge; Preda, Mihai; Cobena, Gregory, 2003 :Adaptive on-linepage importance computation. In :Proceedings of the 12th International Conference on World Wide Web, WWW ’03, pp. 280–290, ACM, New

York, NY, USA,

[2] Białecki, Andrzej, 2012:Nutch search engine. In : White, Tom (ed.)Hadoop: The DefinitiveGuide, pp. 565–579, O’Reilly.

[3] Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C.; Wallach,

Deborah A; Burrows, Mike; Chandra, Tushar; Fikes, Andrew; Gruber,Robert E., 2006:

Bigtable: A distributed storage system for structured data. In:Proceedings of the 7th Conference on USENIX Symposium onOperating Systems Design and Implementation (OSDI ’06), vol. 7, pp.205–218,

[4] Nioche, Julien, 2013,Nutch fight! 1.7 vs 2.2.1. Blog post,

http://digitalpebble.blogspot.co.uk/2013/09/nutch-fight-17-vs-221.html

[5] Peng, Daniel Dabek, Frank, 2010: Large-scale incremental processing using distributed transactions and notifications. In:9th USENIX Symposium on Operating Systems Design and Implementation.

[6] Tom White, "Hadoop: The Definitive Guide", 1st edition, O'Reilly Media, June 2009.

[7] Elasticsearch Server, By Rafal Kuc, Marek Rogozinsk