SWIFT – A Hybrid Search Engine

ENROL NO: **040303**
NAME OF THE STUDENT: **ANKUSH GULATI**
NAME OF THE SUPERVISOR: **MRS. SHIKHA MEHTA**

MAY 2008

Submitted in partial fulfillment of the Degree of
Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAYPEE INSTITUTE OF INFORMATION AND TECHNOLOGY
UNIVERSITY, NOIDA

TABLE OF CONTENTS

Declaration by the Candidate

I hereby certify that the work, which is being presented in the project entitled *"SWIFT – A Hybrid Search Engine"*, in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering, submitted in the Department of Computer Science and Engineering, Jaypee Institute of Information Technology, (Deemed University), Noida is an authentic record of my work carried out during the period from July, 2007 to May 2008 under the supervision and guidance of *Mrs. Shikha Mehta* .

I have not submitted the matter embodied in this project for the award of any other degree or diploma.

Signature of the student

Ankush Gulati

Date:

Certificate

This is to certify that the work titled **"***SWIFT – A Hybrid Search Engine"* submitted by *Ankush Gulati* in partial fulfillment for the award of degree of B.Tech of Jaypee Institute of Information Technology University, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Date: __/__/____ Sign: _____

Signature of Supervisor
Mrs. Shikha Mehta
Sr. Lecturer

Acknowledgement

I hereby grab the opportunity to express my gratitude towards the Computer Science Department, JIIT Noida, for providing me an excellent environment & facilities in order to prepare and present the report of my project.

I sincerely thank *Mrs. Shikha Mehta*, Department Of Computer Science and Engineering, who initially put forth the idea of creating this tool. This project is result of her initiative and constant motivation. I received great inspiration and constant encouragement from all my revered teachers and am highly acknowledged. They all gave me valuable help.

Last but not the least I thank all the concern one's who directly or indirectly helped me in the continuous progress of the project.

Ankush Gulati,
Department of Computer Science,
Jaypee Institute of Information Technology, Noida.

Summary

The large size and the dynamic nature of the Web highlight the need for continuous support and updating of Web based information retrieval systems such as search engines. Due to resource constraints, search engines usually have difficulties in striking the right balance between time and space limitations. In this project, we propose a simple yet effective model for a search engine. We suggest a hybrid design which brings together the best features that constitute a search engine. To give an overview, the whole mechanism of how a search engine works is provided. Further, the model is discussed in detail. We then demonstrate how the proposed model, which embodies features like Fingerprinting, Compressor, Importance number and Refresher can improve the efficiency of a simple search engine if applied on a large scale.

The World Wide Web has grown from a few thousand pages in its early days to more than two billion pages at present. Moreover, the web is not structured at all and finding your desired page on the web without a search engine can be a painful task. That is why; search engines have grown into by far the most popular way for navigating the web.

Engineering a search engine is a challenging task. Search engines rely on massive collections of web pages that are acquired with the help of web crawlers, which traverse the web by following hyperlinks and storing downloaded pages in a large database that is later indexed for efficient execution of user queries. Many researchers have looked at web search technology over the last few years, including crawling strategies, storage, indexing, and ranking techniques as a complex issue. The key to a search engine is, that it needs to be equipped with an intelligent navigation strategy, i.e. enabling it to make decisions regarding the choice of subsequent actions to be taken (pages to be downloaded etc). In this project, we propose an architecture which can be helpful in improving the efficiency of search engines. Our main goal is to improve the quality of web search engines and build an architecture that can search the ever growing web data in a better way.

_____	_____
Signature of Student	Signature of Supervisor
Ankush Gulati	*Mrs. Shikha Mehta*
Date	Date

LIST OF FIGURES

LIST OF ACRONYMS

WWW	World Wide Web
HTTP	Hyper Text Transfer Protocol
ASP	Active Server Page
URL	Uniform Resource Locator
MD5	Message-Digest algorithm 5
SHA	Secure Hash Algorithm
NOF	Number of Forward Links
URI	Uniform Resource Identifier
HTML	Hyper Text Markup Language
IMP	Importance Value
DNS	Domain Name System
W3C	World Wide Web Consortium

Chapter 1

Introduction

1.1. General

SEARCH ENGINES

The World Wide Web has grown from a few thousand pages in its early days to more than two billion pages at present. Moreover, the web is not structured at all and finding your desired page on the web without a search engine can be a painful task. That is why; search engines have grown into by far the most popular way for navigating the web. The large size and the dynamic nature of the Web highlight the need for continuous support and updating of Web based information retrieval systems such as search engines. A search engine is an information retrieval system designed to help find information stored on a computer system, such as on the World Wide Web, inside a corporate or proprietary network, or in a personal computer.

Search Engines have played a fundamental role in making the Web easier to use for millions of people. Its invention and subsequent evolution helped fuel the Web's growth by creating a new way of navigating hypertext: searching. Before search engines, a user who wished to locate information on the Web either had to know the precise address of the documents he sought or had to navigate patiently from link to link in hopes of finding his destination. According to a study at Sun Microsystems, the number of servers on the WWW is doubling every 53 days. If this to be believed then there is an urgent need to develop new applications that realize the power of the Web, but also in the technology needed to scale applications to accommodate the resulting large data sets and heavy loads. I have always been interested in the working mechanism of search engines. So I chose this field as it has a lot of scope in future too.

Engineering a search engine is a challenging task. Search engines rely on massive collections of web pages that are acquired with the help of web crawlers, which traverse the web by following hyperlinks and storing downloaded pages in a large database that is later indexed for efficient execution of user queries. Many researchers have looked at web search technology over the last few years, including crawling strategies, storage, indexing, and ranking techniques as a complex issue. The key to a search engine is, that it needs to be equipped with an intelligent navigation strategy, i.e. enabling it to make decisions regarding the choice of subsequent actions to be taken (pages to be downloaded etc).

How do Search Engines work ?

There are differences in the ways various search engines work, but they all perform three basic tasks:

- They search the Internet -- or select pieces of the Internet -- based on important words. [CRAWLER]
- They keep an index of the words they find, and where they find them. [INDEXER]
- They allow users to look for words or combinations of words found in that index. [SEARCHER]

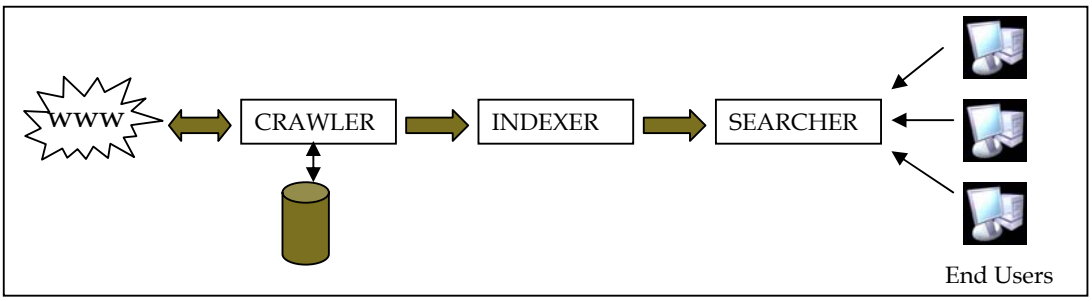


Fig 1:Basic Design of a search engine

Crawler

A web crawler (also known as a web spider or web robot) is a program or automated script which browses the internet seeking for web pages to process. Many applications mostly search engines, crawl websites everyday in order to find up-to-date data.

Indexer

Search engine Indexing entails how data is collected, parsed, and stored to facilitate fast and accurate information retrieval. The indexer module extracts all the words from each page and records the URL where each word occurred.

Here the ranking module has the task of sorting the results such that results near top are the most likely results for the user. The ranking module consists of rank distribution algorithm which assigns a random rank to a web page and then computes the rank of other web pages.

Project as an enquiry

The process of looking at the project as an enquiry whose answer would be the result of our project led to the following result:

How can we enhance the user's decision making ability by efficiently handling the large volumes of information available?

The other minor questions that arise due to the enquiry question:

- ✓ How can a few URLs completely span the web?
- ✓ What do you mean by distributed systems?
- ✓ How would the performance be increased by applying distributed approach to the design?
- ✓ How to make searching user-friendly and user-specific?
- ✓ What features can be incorporated from various existing models and why?
- ✓ What is Fingerprinting?
- ✓ How can we handle redundancy issues?

The process of looking at the project like an enquiry helped in understanding the basic requirements of the project. It also helped in checking what were our goals and how close it is to the goal had set for ourselves.

1.2. Problem Statement

1) <u>Problem Statement:</u>

On the basis of recent studies made on the structure and dynamics of the web itself, it has been analyzed that the web is still growing at a high pace, and the dynamics of the web is shifting. More and more dynamic and real-time information is made available on the web.
Our aim is to design a search engine that meets the challenges of web growth and update dynamics

2) <u>Steps taken to define the problem:</u>

- First of all, extensive literature survey was done to have an idea of the topic – Search Engines and how do they work.
- Most importantly, the applications and implications of the topic were analyzed. For example, constraints like.
- After that, recent work done in this field was analyzed and areas that are yet to be worked upon were explored.
- Finally, our own limitations to develop the project were figured out. For example, we cannot work on large scale systems because of the lack of system requirements.

3)<u>Project Area & justification of project area / problem statement</u>

TC 8 : *Information Systems*

WG 8.3 *Decision Support Systems* **&** WG8.5 *Information Systems in Public Administration*

The aim of the working committee of IFIP standards(WG 8.3 & 8.5) i.e. to promote and encourage interactions among professionals from practice and research and advancement of investigation of concepts, methods, techniques, tools, and issues related to information systems in organizations is well justified by the project domain thus chosen by us. Search Engines have played a fundamental role in making the Web easier to use for millions of people. Its invention and subsequent evolution helped fuel the Web's growth by creating a new way of navigating hypertext: searching. Before search engines, a user who wished to

locate information on the Web either had to know the precise address of the documents he sought or had to navigate patiently from link to link in hopes of finding his destination. As the Web grew to encompass millions of sites, with many different purposes, such navigation became impractical and arguably impossible. According to a study at Sun Microsystems, the number of servers on the WWW is doubling every 53 days. If this to be believed then there is an urgent need to develop new applications that realize the power of the Web, but also in the technology needed to scale applications to accommodate the resulting large data sets and heavy loads. I have always been interested in the working mechanism of search engines. So I chose this field as it has a lot of scope in future too.

The research work in my project is being done in the field of improving the efficiency of search engines, specifically dynamic search engines. Thus after going through many research papers and lectures I understood the need and scope of effective utilization of information technologies in organizational context.

4) <u>Main Problems faced to define the problem:</u>

- There is not a rich body of literature describing practical large-scale crawling and searching systems, and in particular, very few address issues relating to the dynamic web.

- Different search engines follow different techniques and hence, use different metrics to evaluate their performance. Thus, it was difficult to compare and contrast performance results two different search engine results.
- Despite the importance of large-scale search engines on the web, very little academic research has been conducted on them which made it difficult to have a exact idea of how things work and can be further improved.

- Almost all the papers and publications have a limitation. While one used a simplified strategy, the other could not theoretically prove its practical observations.

- With the astronomical growth in the size of WWW, despite the fact that our systems are efficient for the current scenarios, we continuously need to invent new techniques and algorithms to be able to cope with the increasing demands.

Chapter 2

Background Material

2.1 Recent Research Work

We studied a few research works currently going on in the field of search engines out of which a few were quite useful to us in designing our model.

The FAST Search Engine service is running live at www.alltheweb.com and major portals worldwide with more than 30 million queries a day, about 700 million full-text documents, a crawl base of 1.8 billion documents, updated every 11 days, at a rate of 400 documents/second. The FAST Search Engine architecture copes with several of these problems by its key properties. The system architecture is relatively simple, and this makes it manageable even when it grows. In a real-life system with service level requirements, simplicity is crucial to operating the system and to being able to develop it further. Being heterogeneous and containing intelligence with regards to scheduling and query processing makes this a real-life example of dealing with web dynamics issues today.

Another system, a high availability system of crawling called Dominos has been created in the framework of experimentation for French Web legal deposit carried out at the Institute National de l'Audiovisuel (INA). Dominos is a dynamic system, whereby the processes making up its kernel are mobile. 90% of this system was developed using Erlang programming language, which accounts for its highly flexible deployment, maintainability and enhanced fault tolerance.

2.2 Study of a Theoretically Sound Book

The main book that we referred to and studied was **Decision Support Systems and Intelligent Systems** by *Efraim Turban* and *Jay E.Aronson.* This book presents the fundamentals of the techniques with which management support systems are developed.

Generally, the book covers five main subjects: Knowledge Management, Decision Support Systems, Fundamentals of Intelligent Systems, Advanced Intelligent Systems, and Implementation.

Decision Support Systems and Intelligent Agents particularly discuss about Intelligent Software Agents. They define 4 levels of Intelligence. According to those intelligence levels, Search Engine is classified as having Level 1 of intelligence (Level 0 being the lowest).

Search Engines as Intelligent Agents

Most of the agents are single task agents i.e. they perform brute search on internet. Here comes the need of multi-task search agents. Then it defines Search Engines as Intelligent Agents. With these agents at work, the competent user's decision making ability is enhanced with the information.

The book is intended to people like us who actually wish to apply cryptographic methods to their programs, and so the theoretical discussions and mostly at introductory level -- sufficient to make us understand how a search engine acts as an intelligent agent and intelligent algorithm makes search engine work effectively.

The above book was really helpful in making us understand the fundamental and important concepts of intelligent information systems. These books broadened our horizons and had a deep impact on our project.

2.3 Literature Survey

1. Knut Magne Risvik,Rolf Michelsen, *"Search Engines and Web Dynamics*"

   Overview:

   This paper presents several dimensions of web dynamics in the context of large-scale Internet search engines. It shows how the problems arise in all components of a reference search engine model. Furthermore, the FAST Search Engine architecture as a case study for showing some possible solutions for web dynamics and search engines is used. Both growth and update dynamics clearly represent big challenges for search engines. Future evolution of web dynamics raises clear needs for even more intelligent scheduling to aggregate web content as well as technology for push-based aggregation.

2. Monica Peshave, *"How Search Engines Work and A Web Crawler Application"*

   Overview:

   The paper describes in detail the basic tasks a search engine performs. An overview of how the whole system of a search engine works is provided. This paper also lists proposed functionalities as well as features not supported by the web crawler application. This paper explains the anatomy of typical search engine and demonstrates the working of a web crawler developed in Java. It discusses the functionalities of all the components involved in finding information on the Web.

3. Altigran S.da Silva, Eveline A. Veloso, Paulo B. Golgher, Berthier Ribeiro-Neto, Alberto H. F. Laender, Nivio Ziviani, *"CoBWeb"*

   Overview:

   This paper describes CoBWeb, an automatic document collector, whose architecture is distributed and highly scalable. CoBWeb aims at collecting large amounts of documents per time period, while observing operational and ethical limits in the crawling process. These results indicate that by running several collecting processes and by properly distributing it is possible to obtain collection rates comparable to that of the crawlers of commercial search engines. Another contribution can be the good results obtained by the use of the parasite cache approach to update the set of collected documents.

4. Allan Heydon, Marc Najork, *"Mercator: A Scalable, Extensible Web Crawler"*

   Overview:

   This paper describes Mercator, a scalable, extensible web crawler written entirely in Java. It enumerates the major components of any scalable web crawler, comment on alternatives and tradeoffs in their design, and describes the particular components used in Mercator. It also describes Mercator's support for extensibility and customizability. calable web crawlers are an important component of many web services, but they have not been very well documented in the literature. Building a scalable crawler is a non-trivial endeavor because the data manipulated by the crawler is too big to fit entirely in memory, so there are performance issues relating to how to balance the use of disk and memory.

5. Arvind Arasu, *"Searching the Web"*

   Overview:

   This paper offers an overview of current web search engine designs. It shows search engines components covering crawling, page storing and indexing. Furthermore, it emphasizes on the use of link analysis for boosting search performance. The link structure of the web contains a lot of useful information that can be harnessed to support keyword searching.Page Ranking is a global ranking scheme that can be used to rank search results.HITS sets a set of authority pages and a set of hub pages.

6. Sergey Brin and Lawrence Page, *"The Anatomy of a Large-Scale Hypertextual Web Search Engine"*

   Overview:

   The paper describes a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext.This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext.This paper explains simple improvements to efficiency including query caching, smart disk allocation, and subindices. It indicates that PageRank can be personalized by increasing the weight of a user's home page or bookmarks.

7. Behnak Yaltaghian, Mark H. Chignell, *"Effect of Different Network Analysis Strategies on Search Engine Re-Ranking"*

   Overview:

   The research described in this paper examined two different approaches to building the co-citation network that the authors have used in re-ranking the set of results returned by a search engine. These results indicate search based on a network-analytic relevance prediction model significantly improves the precision of results as compared to the Google search engine.

8. Behnak Yaltaghian, Mark H. Chignell, ***"****Re-ranking Search Results using Network Analysis"*

   Overview:

   In the present paper, the author reviews methods of structured search for information on the World Wide Web. The author proposes new methods based on co-citation and network analysis. The authors have tried to see how well search engine rankings can be improved using a combination of co-citation and network analysis under ideal conditions. Using Google as base for study, they have shown that almost all of the network analysis measures can significantly improve on the Google selection

2.3 Literature Integration

On the basis of the literature survey done, the following inferences have been drawn on which our project can be based. To start with, the paper “How Search Engines Work and a Web Crawler Application” gives an overview on how the whole system of a search engine works. The paper describes in detail the basic tasks a search engine performs. Secondly, the paper “Search Engines and Web Dynamics” gives an insight into the web dynamics in the context of large-scale Internet search engines. It further suggests various approaches to improve the efficiency of the search engines like the refreshing algorithm, the heterogeneous architecture. Further, “CoBWeb – A Crawler for the Brazilian Web” presents a crawler with distributed architecture. This paper presents a unique method for resolving the *same page* problem. Furthermore, “Mercator: A Scalable, Extensible Web Crawler”, another crawler design, enumerates the main components required in any scalable crawler, and discusses design alternatives for those components.

Given the current size and high growth rate of the web, a comprehensive web directory may contain thousands of pages in a particular category. In such a case, it might be impossible for a user to look through all the relevant pages within a particular topic in order to identify the ones that best represents the required topic. It would be more time efficient for a user to view the web pages in order of their importance.

A way to solve this problem is to use a ranking function. While these rankings can work well in some cases, they do not capture the fast changing structure of the web. Google uses a algorithm known as Page Ranking algorithm. This algorithm is based on rank distribution model. The intuition of page rank is that a page on the web is important if there are lots of other important pages pointing to it. But this algorithm strongly depends on the presumption that the web is strongly connected. But in reality, the web is far from strongly connected. So these ranking modules need to be modified.

A modified page ranking has been proposed which removes the dependability of this algorithm on the strong connectivity of the web. This modified algorithm works on page-ranking as the base. After the rank has been assigned to all the web pages, it reassigns the rank by decaying the links of the graph. This method of decaying is referred as decay factor which is introduced on the rank calculated by original page ranking algorithm.

2.3 Review

2.3.1 Review of important Research Papers

1. Knut Magne Risvik,Rolf Michelsen, *"Search Engines and Web Dynamics*"

   Inquiry questions:

   - How can we address the issue of web dynamics in the context of large scale internet search engines?
   - How can the heterogeneous architectures prove helpful?

   Process:

   - Web Dynamics and its problems are introduced.
   - FAST Search Engine architecture is used as a case study for showing possible solutions.
   - Discussion on heterogeneous architecture.
   - Finally, it discusses future of the web.

2. Monica Peshave, *"How Search Engines Work and A Web Crawler Application"*

   Inquiry questions:

   - How does the whole system of a search engine work?
   - What functionalities and features are not supported by the crawler application?

   Process:

   - First problem was stated
   - Extensive background was given
   - Presented the anatomy of a large scale Hypertext Transfer Protocol (HTTP) based Web search.
   - The GUI of the application developed helps the user to identify various actions performed.
   - Additionally, a web crawler is developed and implemented in Java.

3. Allan Heydon, Marc Najork, *"Mercator: A Scalable, Extensible Web Crawler"*

   Inquiry questions:

- What is a scalable and extensible web crawler?
- How to balance the use of disk and memory?

Process:

- Problem was stated
- Related work was surveyed.
- The main components of Mercator were discussed.
- The alternatives and tradeoffs in the design were compared.
- Performance measurements and web statistics were reported.

4. Altigran S.da Silva, Eveline A. Veloso, Paulo B. Golgher, Berthier Ribeiro-Neto, Alberto H. F. Laender, Nivio Ziviani, *"CoBWeb"*

Inquiry questions:

- What is an automatic document collector?
- How can we increase our collection rates while observing operational and ethical limits in the crawling process?

Process:

- Explains the fact that using distributed architecture can help us achieve the crawling rates of commercial crawlers.
- Describes the application of parasite cache approach on a set of collected documents.

i. Flowchart of Process Followed In Research Papers by authors

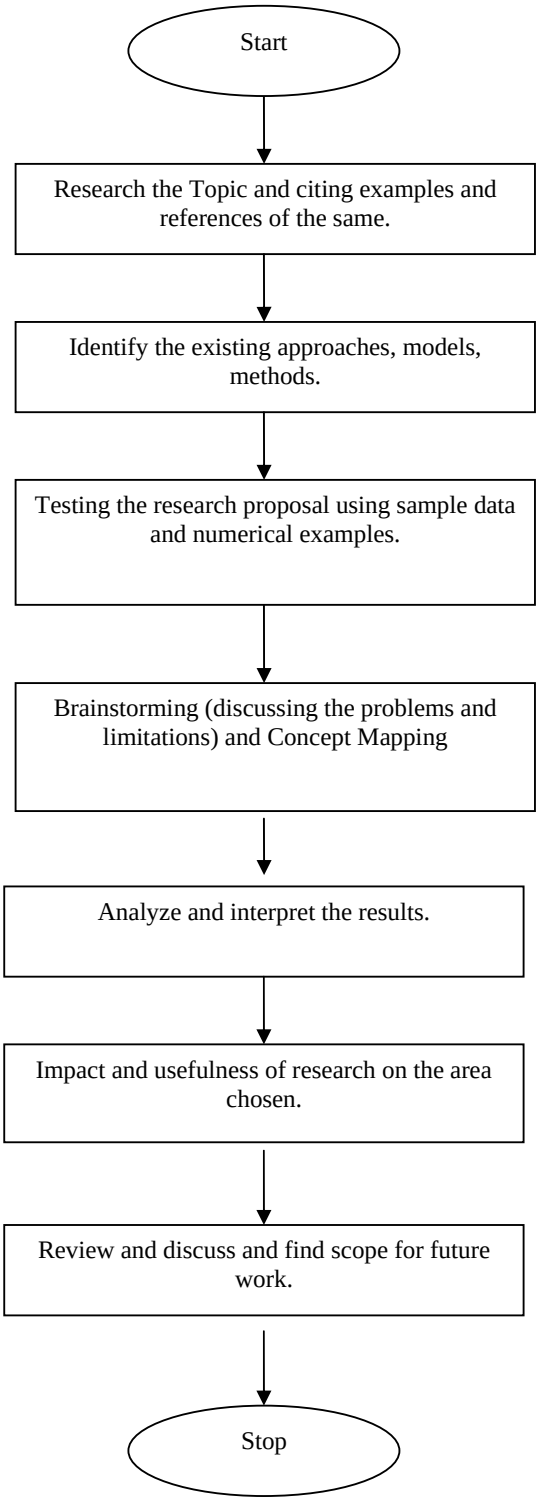


Fig 2

Chapter 3

Contributional Work Descriptions & Results

3.1 Methodology

3.1.1 Flowchart Of Integrated Process Followed

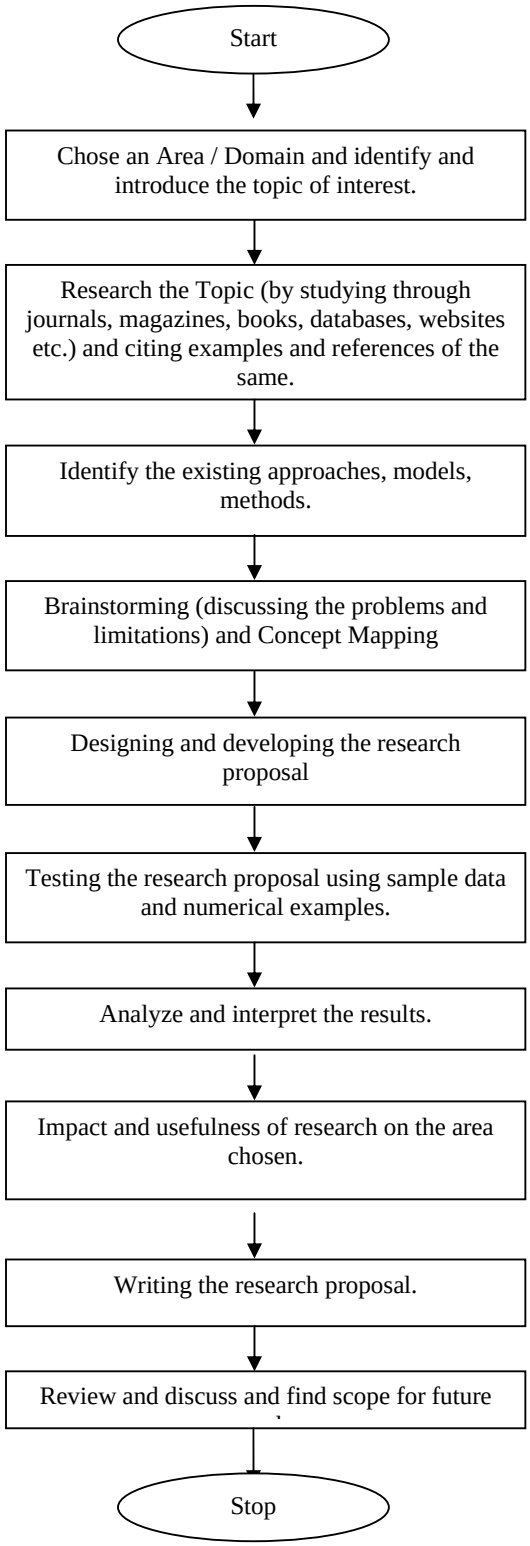


Fig 3

3.1.2 ***How is our Search Engine "Hybrid"?***

On the basis of the study of various designs studied, we have decided to integrate the following features from all the designs in our search engine to make it an efficient search engine :

FAST Crawler

Features included in our search engine:

- Freshness algorithm
- Heterogeneous Crawlers
- Heterogeneous Updation mechanism

COBWeb

Features included in our search engine:

- Inclusion of *"Importance Number"*
- Content based signatures for "Page seen" problem

DOMINOS

Features included in our search engine:

- Distributed Architecture

Mercator

Features included in our search engine:

- Content seen test using fingerprinting

3.2 Modeling, Analysis and Work plan

3.2.1 *Modelling :*

Sequence Diagram

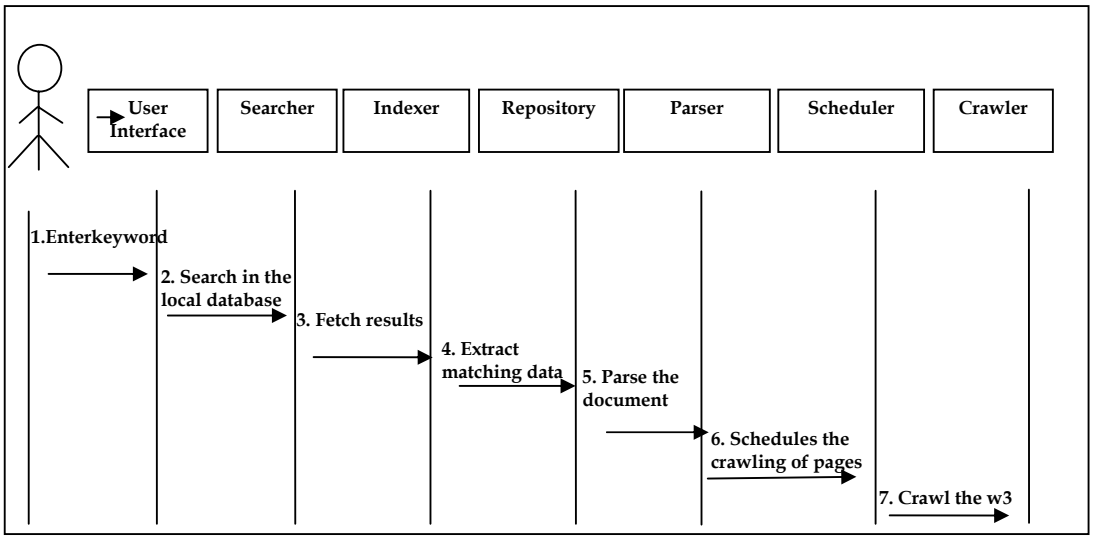


Fig 4

Activity Diagram

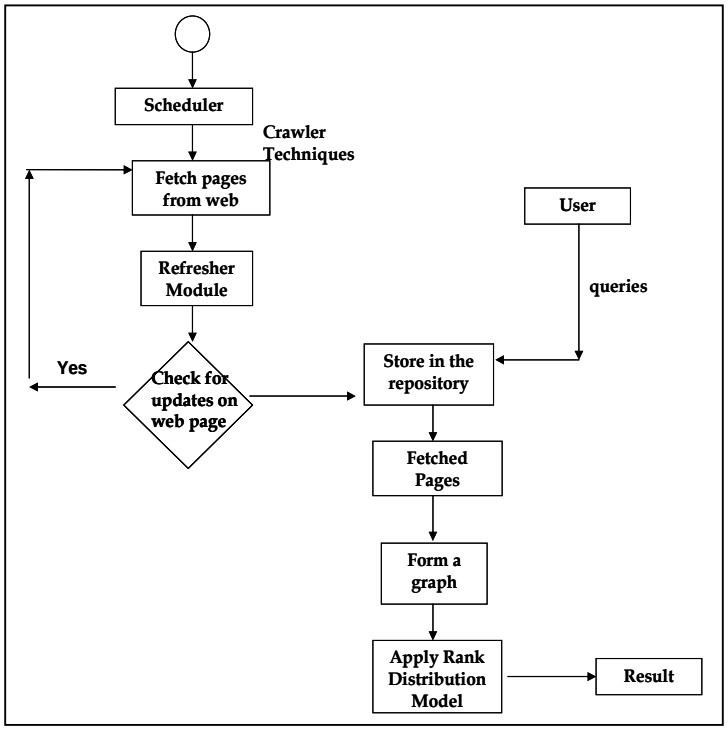


Fig 5

3.2.2 *Analysis:*

On the basis of the literature survey, it was analyzed that there are different ways to improve the performance of web search engines.

There are three main directions:

1. Improving user interface on query input.
2. Using Filtering towards the query results.
3. Solving algorithms in web page spying and collecting, indexing, and output.

And, method 3 is the fundamental solution for any direct search engines to deal with the problems of unequal accessing, out-of-date information, and low metadata using, as well as information coverage.

The field of search engines is infinitely vast; there is a never ending scope for improvement. Hence, we will follow a simple approach that tries to address all the key issues with optimal and effective solutions to attain the desired results.
The overall tool will be developed in an incremental fashion with all the functionalities divided into separate modules which will be developed as per the work plan. All the functionalities will be developed following the approach mentioned in the report that i.e. by using the different strategies for different modules on the basis of literature survey.

It was also seen that there is no solution for Crawler traps where in the crawler is trapped in an infinite loop accessing the same site. It is done by the site owners to increase there search statistics and spammer sites. No automatic technique to handle this problem was found. It can be human handled only.

3.2.3 Work Plan:

S.No	Phase
1.	Literature Survey
2.	Analysis of different algorithm designs.
3.	Designing
4.	Implementation :Crawler
5.	Implementation :HTML Parser
7.	Implementation :User Interface
8.	Implementation :Searcher
9.	Implementation :Indexer
10..	Implementation :Page Refresher
11..	Implementation :Distributed System
11.	Integration : Merge all modules together
12.	Testing & Debugging
13.	Final Report

3.3 Design, Implementation, Testing

3.3.1 Design:

Application Design

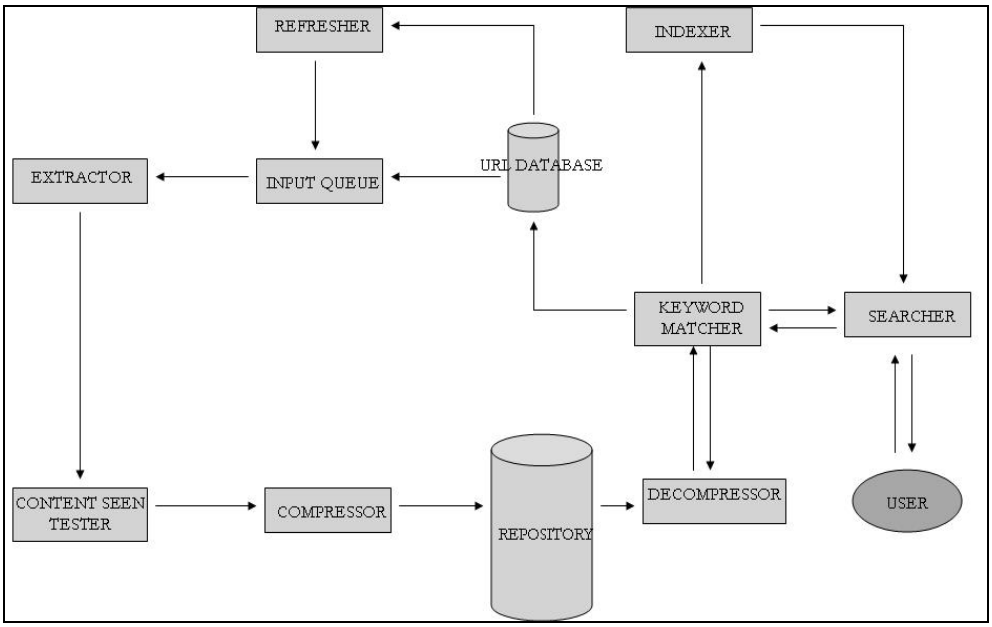


Fig 6: Design of the Hyrbrid model

3.3.2 *Implementation*

A. Data Flow among modules

The *Initial_links table* provides a list of seed URLs as the initial input and the *Crawler* module repeatedly executes the following steps. Pick a URL from the table, download the corresponding document, and extract any links contained in it. With the help of the *Parser* module, it is ensured that no extracted file is encountered before and there does not exist a copy of the same in the *Repository* by applying *Content Seen Testing*. After the document has passed the *Content Seen Test*, its *Fingerprint* is saved in the *Links* table along with the *number of its forward links*. The document is then sent to the *Compressor* module which compresses the document and stores it in the *Repository*.

Alongside, the *Refresher* module works on refreshing the populated Repository. It takes the already visited URLs from the *Links* table and downloads them. The *Refresher* then applies

the refreshing algorithm with the help of Fingerprinting mechanism and updates the local Repository with the fresh copies of existing URLs. In the background, another module namely, the Indexer generates a graph of resulting documents and calculates a rank for each document using the Page Ranking algorithm.

On the other hand, when a user searches for a specific keyword(s), the *Searcher* module fires a query to the Keyword Matcher. The *Keyword Matcher* requests the *Decompressor* to decompress the documents stored in Repository one by one. It searches for the requested keyword(s) within each page it gets from the *Decompressor*. The *Sorter* further sorts the resulting documents on the basis of their rank as well as the I*mportance Number*. The Searcher fetches the results from the *Sorter* and displays the results in hyperlinked format. Finally, for all the links that are accessed by the User, an update is sent by the Searcher to the URL database to increment its *Importance Number*.

B. Platform

Our application runs on Microsoft Visual Studio.NET 2005 with Oracle 10g as backend.

C. Other Qualities:

a. Usability

The ease with which our software can be used by people for searching the web without any computer background.

b. Scalability

Execution time should remain constant with increase in number of users (processors).

3.3.3 Testing

A. Efficiency:

We have to test that the execution time in the best case. It should be less than execution time in the worst case.

INDEXER :

Best Case

When the most populated link exists at the top of the database.

Worst case:

When the most populated link exists at the bottom of the database.

Average case:

When the most populated link exists at an intermediate node.

REFRESHER ***:***

Best Case

When the page has not been changed/updated.

Worst case:

The page no longer exists as it is not hosted on the web now.

Average case:

When pages are changed/updated, their new key sequence has to be generated and they have to be saved in the repository.

*CRAWLER****:***

Best Case

When initial seeds given are densely populated, i.e. they contain more number of links.

Worst case:

When initial seeds given sparsely populated, i.e. they contain less number of links

Average case:

None

B. Accuracy:

1. Incorrect type of links fetched (different file formats, URL rewritten links).
2. Incorrect graph being generated by the indexer.
3. Incorrect information being fetched by the parser.
4. Incorrect page rank.

C. Limitations:

1. Will not work on any other platform other than Windows.
2. Will interpret only a few file formats not all.

D. Test Plan

S.No.	Type Of Testing	Things to be Tested
1	Unit/Module Testing	*Crawler*
		Test whether seed URLs are valid or not.
		Test whether correct links are being extracted.
		Test that the extracted links are correctly stored in the database.
		Test the code for varying load conditions (vary the number of seed URLs).
		Parser
		Test whether the web page is perfectly downloaded and its body is correctly extracted from it.
		Test the calculated values of forward_links for each link.
		Test the fingerprint (MD5 Key generation) and storage for each webpage.
		Test the compression method by matching the data with its decompressed form.
		Refresher
		Testing the correct access of fingerprint from the database.
		Testing the perfect matching of previous and fresh fingerprints.
		Indexer
		Test the correct formation of graph from the unique links stored in the graph.
		Test the perfect calculation of rank for each node.
		Testing of the Page Rank algorithm.
		Searcher
		Test the time efficiency of the searcher by using different keywords (common and rare).
		Test the layout consistency for all cases (for all sizes of output).

2	Integration Testing	Test whether correct URLs are passed to the Parser module.
		Test whether the Parser correctly updates the database with the new fields.
3	System Testing	Testing of Oracle Database connection.
		Testing of connected systems executing different modules.
		Testing the effective working of heterogeneous crawling distributed on various systems.
		Testing the effective working of heterogeneous refreshing distributed on various systems.
		Testing data consistency while multiple machines access the same database.
		Testing the performance under varying load conditions (distributed and serial computing).
4	User Acceptance Testing	Test the search result relevance.
		Test the output layout.
		Test the working of hyperlinked results.
		Testing the performance by increasing the load (accessing the search engine from multiple machines).

3.3.4 Results

Functionality of various modules

1. CRAWLER

- Extract (Findlinks.cs): Extracts pages from the web using the seed links stored in initial_links table.
- Store (Start.cs): Stores the new URL in Links table.

Url	Depth
http://www.indiamobiles.com/	1
http://www.mobiles.in/	1
http://www.w3schools.com/	1
http://www.screenindia.com/	1
http://www.raaga.com/	1
http://www.freeprogrammingresources.com/	1
http://www.hindisong.com/	1
http://www.imdb.com/	1
http://www.rediff.com/movies/index.html	1
http://www.cnn.com/WORLD/	1
http://www.freeprogrammingresources.com/	1
http://www.sun.com/java/	1
http://www.leepoint.net/notes-java/index.html	1
http://www.cprogramming.com/	1
http://www.cricinfo.com/	1
1 - 15	

Fig 7

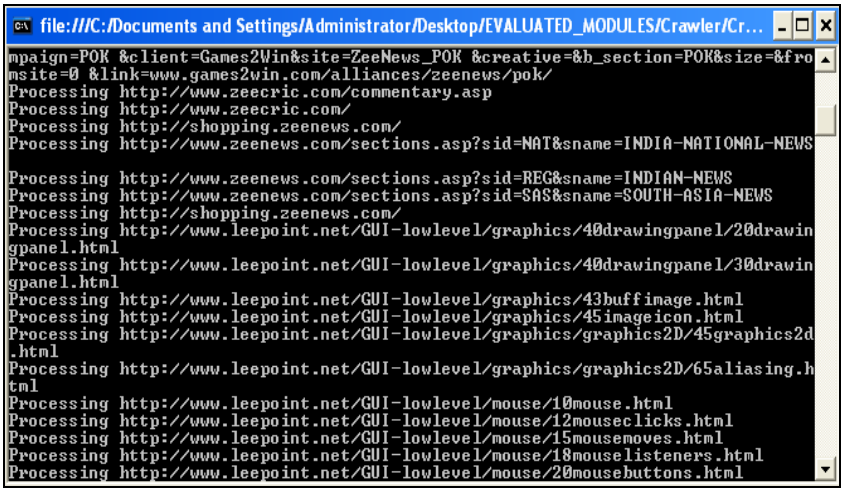
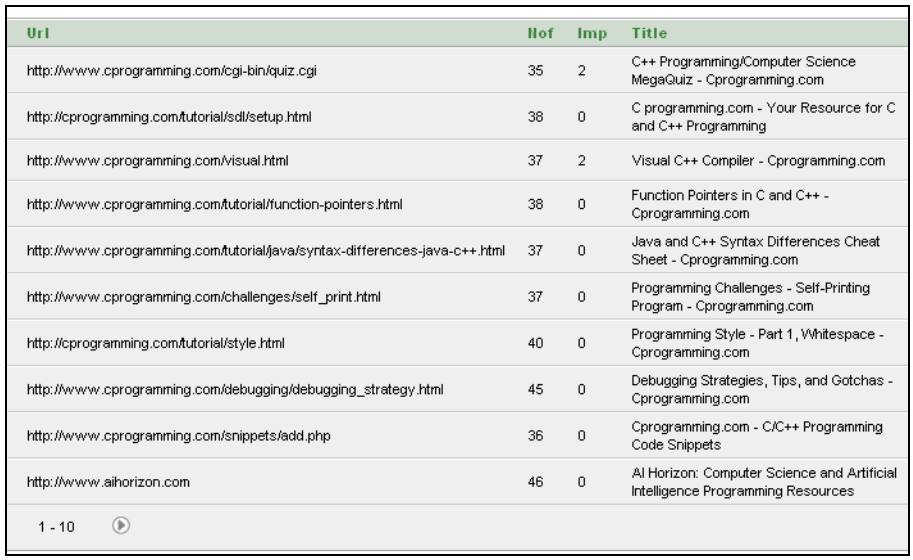


Fig 8

2. PARSER

- Content Seen Testing (Fingerprint.cs): Matches the contents of the downloaded page with the existing pages to reduce data redundancy.
- Compressor-Decompressor (Compression.Cs): Compresses & decompresses the non-redundant pages on specific points.



Url	Nof	Imp	Title
http://www.cprogramming.com/cgi-bin/quiz.cgi	35	2	C++ Programming/Computer Science MegaQuiz - Cprogramming.com
http://cprogramming.com/tutorial/sdl/setup.html	38	0	C programming.com - Your Resource for C and C++ Programming
http://www.cprogramming.com/visual.html	37	2	Visual C++ Compiler - Cprogramming.com
http://www.cprogramming.com/tutorial/function-pointers.html	38	0	Function Pointers in C and C++ - Cprogramming.com
http://www.cprogramming.com/tutorial/java/syntax-differences-java-c++.html	37	0	Java and C++ Syntax Differences Cheat Sheet - Cprogramming.com
http://www.cprogramming.com/challenges/self_print.html	37	0	Programming Challenges - Self-Printing Program - Cprogramming.com
http://cprogramming.com/tutorial/style.html	40	0	Programming Style - Part 1, Whitespace - Cprogramming.com
http://www.cprogramming.com/debugging/debugging_strategy.html	45	0	Debugging Strategies, Tips, and Gotchas - Cprogramming.com
http://www.cprogramming.com/snippets/add.php	36	0	Cprogramming.com - C/C++ Programming Code Snippets
http://www.aihorizon.com	46	0	AI Horizon: Computer Science and Artificial Intelligence Programming Resources

1 - 10

Fig 9

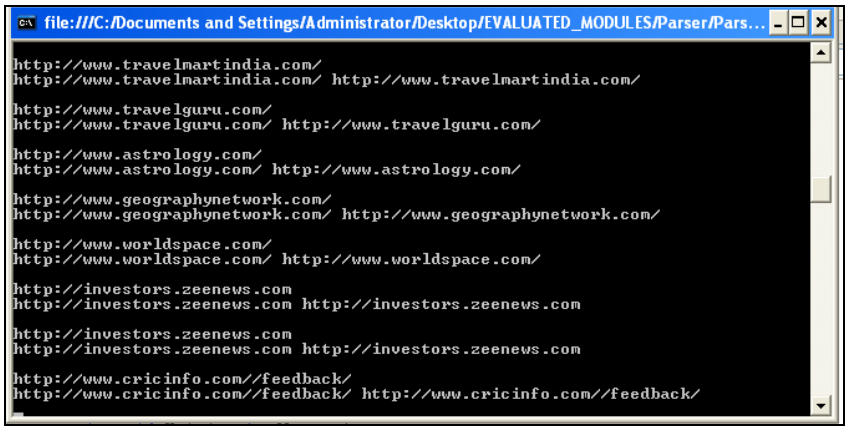


Fig 10

3. SEARCHER

- User Interface: Fires a query to look for the searched keyword in all the files stored in the local repository.

Fig 11

Fig 12

4. INDEXER

- Compute.cs: Sorts the results found on the basis of a rank distribution algorithm.

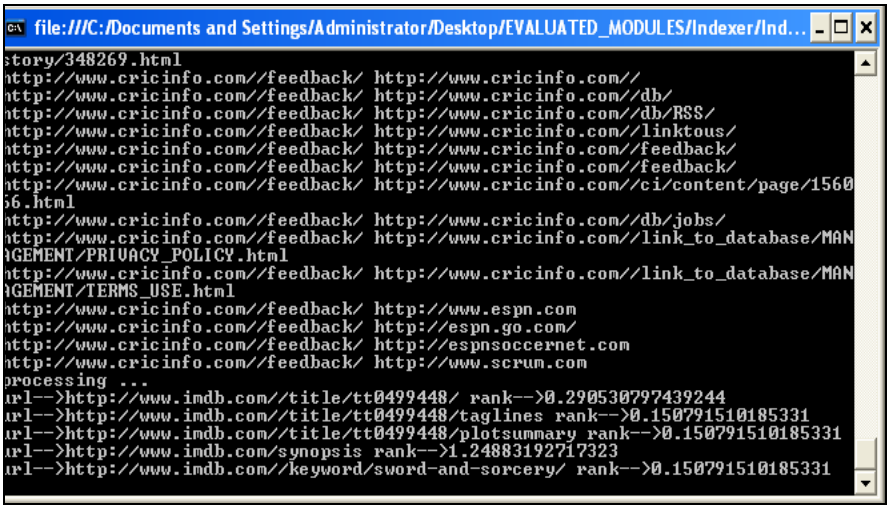


Fig 13

E1	E2
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/title/tt0499448/taglines
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/title/tt0499448/plotsummary
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/synopsis
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/keyword/sword-and-sorcery/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/keyword/prince/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/keyword/second-part/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/keyword/narnia/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/keyword/sequel/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/title/tt0499448/keywords
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/#comment
http://www.imdb.com/title/tt0499448/	http://pro.imdb.com/r/legacy-comaindetails/
http://www.imdb.com/title/tt0499448/	http://pro.imdb.com/r/legacy-compfilmo/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/name/nm1602660/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/name/nm1602660/
http://www.imdb.com/title/tt0499448/	http://www.imdb.com/character/ch0004978/
	row(s) 1 - 15 of more than 500

Fig 14

5. REFRESHER

- Content Matching (Refresh.cs): Updates the local database with fresh copies of web pages from time to time.

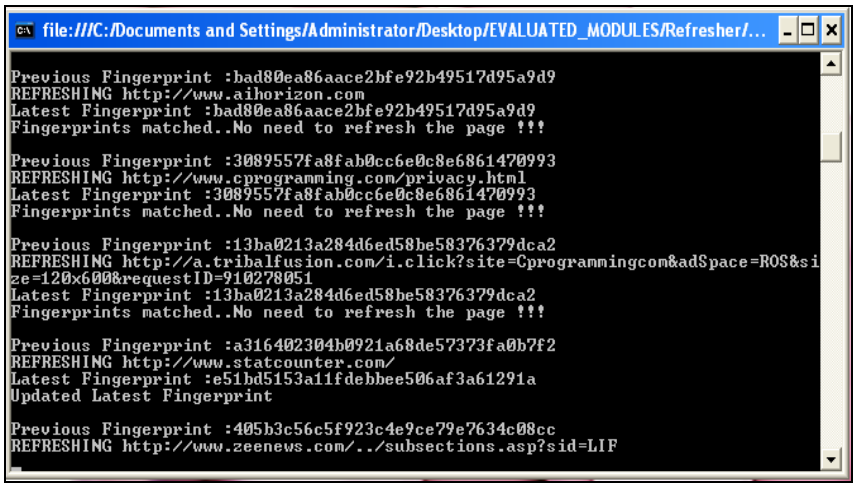


Fig 15

CHAPTER 4

FINDINGS AND CONCLUSIONS

4.1 Findings

EVALUATION MATRIX

Evaluation Parameters	**Our Project**	**FAST**	**MERCATOR**
Crawler Type	Heterogeneous	Non Heterogeneous	Non Heterogeneous
Check pointing	Yes	No	Yes
Database	Compressed	Not Compressed	Not Compressed
Data Redundancy	Nil	Nil	To an extent
Refreshing policy	Non redundant	Redundant	Non redundant
Crawler Traps	No	No	Yes

Function Points

Information Domain Value	Count	Simple	Average	Complex	**Points**
External Inputs(EIs)	2	**3**	4	6	6
External Outputs(Eos)	3	4	**5**	7	15
External Inquiries(EQs)	1	**3**	4	6	3
Internal Logical Files(ILFs)	1	**7**	10	15	7
External Interface Files(EIFs)	2	**5**	7	10	10

Count total 41

FP = Count Total *[0.65 + (0.01* 46)]

= 41* [0.65 + (0.01* 46)]

= 45.51

Design Structure Quality Index

S1= total number of modules defined in the program architecture

S2= number of modules whose correct function depends on the source of data input or that produce data to be used elsewhere.

S3= number of modules whose correct function depends on prior processing

S4= number of database items

S5= number of unique database items

S6= number of database segments

S7= number of modules with a single entry and exit

We have,

S1=5 S5=more than 3000

S2=3 S6=around 1200

S3=4 S7=1

S4= more than 70000

D1=5

D2 = 1- (S2/S1) = (1 - (3 / 5)) = 0.4

D3 = 1- (S3/S1) = (1 - (4 / 5)) = 0.2

D4 = 1-(S5/S4) = (1 - (3000 / 70000)) = 0.95

D5 = 1-(S6/S4) = (1 - (1200 / 70000)) = 0.98

D6 = 1-(S7/S1) = (1- (1 / 5)) = 0.8

DSQI = sigma (Wi Di)

=0.167(5+0.4+0.2+0.95+0.98+0.8)

=0.167*8.33

=1.3911

Halsted Program value

Halsted's theory of "software science" proposed for computer software. The measures are:
n1 = the number of distinct operator that appear in the program = 30
n2= the number of distinct operands that appear in the program= 59
N1= the total number of operator occurrence = 56
N2= the total number of operand occurrences= 102

Halsted length

N = n1log2n1 + n2log2n2
Program value

V = N log2 (n1+n2)

= 4154.09

Volume ratio

L = 2/ n1 * n2/ N2

= 0.0385

Halsted effort

PL = 1 / [(n1/2) * (N2/n2)]
= 0.01960

e= V / PL
= 4154.09 / 0.01960
= 211943.367

COCOMO - II effort

COCOMO II application composition model uses object points – an indirect software measure that is computed using counts of number of

1. Screens(at the user interface)
2. Reports
3. Components

Likely to be required to build the application.

OBJECT TYPE	Count	Simple	Medium	difficult
Screen	1	1	2	**3**
Report	1	2	**5**	8
Components	5			**10**

Total object point = 1*3+1*5+5*10=58

NOP = (object point) * [(100 - % reuse)/100]

= 58 * [(100 – 75)/ 100]

= 58 * 1 /4

= 14.5

For calculation of productivity rate, using the following table

Highlighting the values of our system

Developer's Experience	Very low	Low	Nominal	**High**	Very high
Environment maturity/ capability	Very low	Low	Nominal	**High**	Very high
PROD	4	7	13	**25**	50

So, PROD = 25

Estimated effort = nop / prod = 14.5/ 25 = 0.58

4.2 Conclusion

We have designed a fully distributed, scalable, incremental and extensible model for a search engine. The application has been successfully implemented on a distributed architecture. We have proposed a hybrid model for a search engine as a part of our research work. It copes with several problems faced by existing models with the help of its key properties like fingerprinting, importance value, compression- decompression, smart refresher technique and ranking algorithm all of which have been implemented in a distributed environment.

4.3 Future Scope

We can modify the design and add another dimension to the search engine, the lexical chains; this will allow more efficient searching. The lexical chains contain the list of keywords found in the file along with their count. So, lexical chain analysis provides easy and fast searching. We can optimize the output by using smarter searching on the large database. We can transform the keyword searcher into a broader searcher which can search other file formats as well. Other promising direction is to apply other sources of information, like query logs and click streams, for improving Web searching. We can study more sophisticated text analysis techniques (such as Latent Semantic Indexing) and explore enhancements for them in a hyperlinked setting. Another area of further research is implementing inverted files for every web page, which helps in assigning weights to specific HTML tags and thereby improving the web search.

REFERENCES

i. Qiang Zhu , "An Algorithm OFC For The Focused Web Crawler" *In Proceedings of the Sixth International Conference , Hong Kong*, Aug .2007

ii. Knut Magne Risvik , Rolf Michelsen, "Search Engines and Web Dynamics", *Computer Networks*, 39:289–302, 2002.

iii. Qingzhao , Tan,Prasenjit Mitra , C.Lee Giles, "Designing Clustering-Based Web Crawling Policies for Search Engine Crawlers", *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*,2007

iv. Altigran S. da Silva , Eveline A. Veloso , Paulo B. Golgher "CoBWeb – A Crawler for the Brazilian Web", *String Processing and Information Retrieval Symposium*, 1999 **.**

v. Vladislav Shkapenyuk , Torsten Suel " Design and Implementation of a High-Performance Distributed Web Crawler ", *Proceedings 18th International Conference***,**2002.

vi. Younes Hafri , Chabane Djeraba " Dominos : A New Web Crawler's Design ",*IWAW* 2004.

vii. Allan Heydon " Marc Najork " Mercator : A scalable , Extensible Web Crawler" A. Heydon and M. Najork, "Mercator: A scalable, extensible web crawler," *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.

viii. Arvind Arasu , Junghoo Cho , Hector Garcia-Molina , Andreas Paepcke , Sriram Raghavan "Searching the Web". *ACM Transactions on Internet Technology*, 2001.

ix. Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", *Computer Networks and ISDN Systems***,** Volume 30 , Issue 1-7 (April 1998) Pages: 107 - 117 ISSN:0169-7552 *1998.*

x. Michelangelo Diligenti , Marco Gori , Marco Maggini " Web Page Scoring Systems for Horizontal and Vertical Search",ACM ,2005.

Appendix A

GLOSSARY

A

Architecture-- Information architecture (IA) is the art and science of expressing a model or concept for information used in library systems, web development, user interactions, database development, programming, technical writing, enterprise architecture, critical system software design and other activities that require explicit details of complex systems.

C

Crawler-- A web crawler is a program or automated script which browses the World Wide Web in a methodical, automated manner.

D

Database-- A computer database is a structured collection of records or data that is stored in a computer system. A database relies upon software to organize the storage of data.

Distributed Architecture-- facilitates designing for flexibility because distributed systems can often be easily reconfigured by adding extra servers or clients and clients and servers can be developed by competing organizations, giving the customer a choice

F

Fingerprinting

E

Extractor

H

Hybrid

I

Indexer

Information Retrieval-- Information retrieval (IR) is the science of searching for information in documents, searching for documents themselves, searching for metadata which describe documents, or searching within databases, whether relational stand-alone databases or hypertextually-networked databases such as the World Wide Web.

M

MD5 algorithm

Meta Tags-- Meta elements are HTML or XHTML elements used to provide structured metadata about a web page. Such elements must be placed as tags in the head section of an HTML or XHTML document. Meta elements can be used to specify page description, keywords and any other metadata not provided through the other head elements and attributes.

R

Refresher--

Refresh Policy

S

Searcher

Spider-- A spider(also known as web crawler) is a program or automated script which browses the World Wide Web in a methodical, automated manner.

Seeds

T

Title

U

URI-- A URI, or Uniform Resource Identifier, is the physical location of a file on the system.

URL-- A URL, or Uniform Resource Locator, is the location of a file on the Web.

W

World Wide Web-- The World Wide Web (commonly shortened to the Web) is a system of interlinked hypertext documents accessed via the Internet.

Appendix B

DECISION CATALOGUE

In the process of the development of our project we had the following decisions to make:

S.No.	Decision taken	Reason
1.	Choosing the project area/domain i.e. *E-commerce and management of Information Systems*	This particular domain was chosen because it was our area of interest. Moreover, there is always a scope of improvement in the ever growing web and we thought we would be able to contribute to this domain by designing and developing our own architecture.
2.	Choosing the subject in which we would design an application- *Web based intelligent information retrieval systems*	A large number of analyses have been made on the structure and dynamics of the web itself. Conclusions are drawn that the web is still growing at a high pace, and the dynamics of the web is shifting. More and more dynamic and real-time information is made available on the web. The dynamics of the web creates a set of tough challenges for retrieval of information and hence, makes the field interesting.
3.	Narrowing down to a specific problem/area in Information Retrieval systems i.e. *Search engines*	The web is growing in size at an exponential rate. This poses a serious challenge of scalability for search engines that aspire to cover a large part of the web. Thus, we decided to take up the task of developing a smart model for a search engine.
4.	Deciding on our problem statement: Developing a *hybrid model* for a search engine which improves the efficiency of web searching.	After doing the literature survey, we could assimilate that there are various methods that can improve given systems. We decided to incorporate some of them into our design and develop a hybrid model from it.
5.	Selection of specific techniques from existing models- *Fingerprinting , Page-Rank , Refresher*	Various research models use different techniques to handle data redundancy issues. We found Fingerprinting to be an effective way of preventing redundancy completely.
6.	Selection of specific techniques	This feature is not available in any of the models

	from existing models- *Compressor - Decompressor*	we studied but we found it efficient as it could solve the problem of space complexity.
7.	Selection of specific techniques from existing models- *Refresher*	We selected a simple yet powerful algorithm from one of the research works we studied. It was important to select a good algorithm because of the importance of refresher module in a search engine.
8.	Selection of specific techniques from existing models- *Page-Rank*	We opted for Page Rank as it is a link based algorithm and is thus more precise as compared to HITS which is content based.
9.	Division of the project into modules	This was done to make our work more organized and efficient. We followed a step by step process to implement the design.
10.	Deciding on the search horizon- whether to work on global internet or the local intranet	Even though, it would have been much easier searching on a limited set of data i.e. intranet but it could not have answered our primary question of intelligent information retrieval from a dynamic system.
11.	Deciding on indexing algorithm- *Page-Ranking algorithm*	We chose page-Ranking algorithm to index our web pages because it is a link based algorithm which rates the web pages on the basis of its popularity content.
12.	Deciding on the database- *Oracle*	Oracle is the choice of industry all over the world. Moreover, it is very user friendly as well.
13.	Deciding on the platform- *.NET*	We chose .NET because it provides many libraries and API's for web related tools. It is relatively easy to use and working on it gave us an exposure to a new platform.
14.	Deciding on initial seeds.	From various research papers , we concluded that Initial seeds are chosen on the basis of search domain , more the no of outward flowing links from a website as a whole (not webpage) more it will aid the crawler in its search
15.	Distributing the application on a	This decision was made because we discovered our

	distributed system	limitation with system performance. Such huge amount of data and such complex applications were not running efficiently on our system. So, distributing clearly helped address this problem.

Appendix C

TEST REPORT

TEST CASES

Test Case 1:

Date: 24th Jan 2008

System/Module: Crawler Environment:C#.NET/Oracle 10g

Function: Returns the hyperlinks connected to a given webpage.

Pass/Fail: Pass

Entrance Criteria for This Test:

Correct initial seeds at the start of crawler.

Data required to execute the Test:

- Initial Seeds
- Depth

Conditions to Test:

- HTML page exists at the given URL..

Steps to Perform:

- Pass the Initial link to the function.
- Search for hyperlinks on the given web page.
- Save those links in database.

Expected Results:

- The function should return the links hyperlinked to the given web page..

Actual Results:

The index values in the arraylist pos1 were same as those that were observed while making the trees manually.

Test Case 2:

Date: 28h Feb 2008

System/Module: HTML Parser **Environment:C#.NET / Oracle 10g**

Function: Returns the title, no. of forward links, meta tags, bit sequence of a web page.

Pass/Fail: Pass

Entrance Criteria for This Test:

HTML page exists at the given url.

Data required to execute the Test:

- URL

Conditions to Test:

- Whether the extracted links are legal or not.

Steps to Perform:

- Call the function using the object of main class.
- Pass the URL of a web page as input.

Expected Results:

- The function to store the no. of forward links, title, meta tags, bit sequence in the *LINKS* table of the database.

Actual Results:

The function stores the no. of forward links, title, meta tags, bit sequence in the *LINKS* table of the database.

Test Case 3:

Date: 20h March 2008 **Environment:C#.NET/Oracle 10g**

System/Module: Searcher

Function: Returns the links matching to the user query **Pass/Fail: Pass**

Entrance Criteria for This Test:

Succesful execution of Parser module..

Data required to execute the Test:

- Search query

Conditions to Test:

- .Data Reader reading the link should not be null.
- Value of sequence length should be an integral value.

Steps to Perform:

- Take the input keyword from the user.
- Decompress the data and search the keyword in the decompressed data..

Expected Results:

- The links matching the user query are shown to the user.
- The links are shown in the order of their ranking and uswer relevance.

Actual Results:

The expected values were obtained.

Test Case 4:

Date: 20h April 2008 **Environment:C#.NET/Oracle 10g**

System/Module: Indexer

Function: Makes a graph of all the web pages and calculates the rank of all the web pages using rank-distribution model. **Pass/Fail: Fail**

Entrance Criteria for This Test:

All the links have been stored in the database.

Data required to execute the Test:

- All the links from all the web pages.

Conditions to Test:

- Links in the *LINKSTORE* table.
- ArrayList of input values should not be empty.
- Graph should not be empty.

Steps to Perform:

- Create a graph of all the links.
- Evaluate the forward and the backward links of all the web pages.
- Calculate the rank on the basis of forward and backward links.

Expected Results:

- Ranks of all web pages using the rank-distribution model.

Actual Results:

The function did not compute the rank of all the pages.

Test Case 5:

Date: 20h April 2008 **Environment:C#.NET/Oracle 10g**

System/Module: Refresher

Function: Checks for any updations on the web and if any updations found , it replaces the existing page with the updated page. . **Pass/Fail: Fail**

Entrance Criteria for This Test:

Parser should have computed the key for all the web pages.

Data required to execute the Test:

- *LINKS* table.

Conditions to Test:

- Parser module should have computed the MD5 key for all the web pages.
- The web page still exists on internet.

Steps to Perform:

- Compute the key for the web page.
- Match with the existing key if it does not matches , downloads the page and updates it in the repository.

Expected Results:

- Pages should be refreshed if they contain any updations.

Actual Results:

Pages were being refreshed but were not being downloaded and updated in the database.

USER ACCEPTANCE TESTING

User Feedback

The final application was provided to the common users (students) and there feedback was taken on how they found the application. About 35-40 users used the search engine and found the results relevant enough. They were asked to fill a feedback form out of which some are attached with the report as well.

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

USER ACCEPTANCE TESTING

SWIFT : Hybrid Search Engine

Feedback Form

Name :

Designation/ Enrollment No :

Date :

Performance Evaluation Table

(Tick only one of the following)

Parameter	*Poor*	*Average*	*Good*	*Excellent*
Speed				
Relevance of Output				
No. of links generated				
Layout/ Design				
User friendliness				

COMMENTS (if any):

Appendix D

Tools

1. Microsoft Visual Studio .NET Framework 2.0

The Microsoft .NET Framework is a component of the Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering, and is intended to be used by most new applications created for the Windows platform.

The pre-coded solutions form the framework's class library and cover a large range of programming needs in areas including the user interface, data access, cryptography, numeric algorithms, and network communications. The functions of the class library are used by programmers who combine them with their own code to produce applications.

Programs written for the .NET framework execute in a software environment that manages the program's runtime requirements. This runtime environment, which is also a part of the .NET framework, is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine, so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security guarantees, memory management, and exception handling.

The class library and the CLR together comprise the .NET framework. The framework is intended to make it easier to develop computer applications and to reduce the vulnerability of applications and computers to security threats.

1.1 Important features of .NET Framework

1. **Interoperability** - Because so many COM libraries have already been created, the .NET Framework provides methods for allowing interoperability between new code and existing libraries.
1. **Common Runtime Engine** - Programming languages on the .NET Framework compile into an intermediate language known as the Common Intermediate

Language, or CIL; Microsoft's implementation of CIL is known as Microsoft Intermediate Language, or MSIL. In Microsoft's implementation, this intermediate language is not interpreted, but rather compiled in a manner known as just-in-time compilation (JIT) into native code. The combination of these concepts is called the Common Language Infrastructure (CLI), a specification; Microsoft's implementation of the CLI is known as the Common Language Runtime (CLR).

2. **Language Independence** - The .NET Framework introduces a Common Type System, or CTS. The CTS specification defines all possible data types and programming constructs supported by the CLR and how they may or may not interact with each other. Because of this feature, the .NET Framework supports development in multiple programming languages. This is discussed in more detail in the .NET languages section below.
3. **Base Class Library** - The Base Class Library (BCL), sometimes referred to as the Framework Class Library (FCL), is a library of types available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions such as file reading and writing, graphic rendering, database interaction, XML document manipulation, and so forth.
4. **Simplified Deployment** - Installation and deployment of Windows applications has been the bane of many developers' existence. Registry settings, file distribution and DLL hell have been nearly completely eliminated by new deployment mechanisms in the .NET Framework.
5. **Security** - .NET allows for code to be run with different trust levels without the use of a separate sandbox.

The design of the .NET framework is such that it supports platform independence. That is, a program written to use the framework should run without change on any platform for which the framework is implemented. At present, Microsoft has implemented the full framework only on the Windows operating system. Microsoft and others have implemented portions of the framework on non-Windows platforms, but to date those implementations are not widely used.

1. Oracle 10g

An Oracle database system comprises at least one instance of the application, along with data storage. An instance comprises a set of operating-system processes and memory-structures that interact with the storage. Typical processes include PMON (the process monitor) and SMON (the system monitor).

The Oracle RDBMS stores data logically in the form of tablespaces and physically in the form of data files. Tablespaces can contain various types of memory segments; for example, Data Segments, Index Segments etc. Segments in turn comprise one or more extents. Extents comprise groups of contiguous data blocks. Data blocks form the basic units of data storage. At the physical level, data-files comprise one or more data blocks, where the block size can vary between data-files

Oracle database management keeps track of its computer data storage with the help of information stored in the SYSTEM tablespace. The SYSTEM tablespace contains the data dictionary — and often (by default) indexes and clusters. (A data dictionary consists of a special collection of tables that contains information about all user-objects in the database). Since version 8i, the Oracle RDBMS also supports "locally managed" tablespaces which can store space management information in bitmaps in their own headers rather than in the SYSTEM tablespace (as happens with the default "dictionary-managed" tablespaces).

Appendix E

Gantt Chart

	Task Name	Duration	Start	Finish
1	Domain Selection	6 days	Wed 7/25/07	Wed 8/1/07
2	Project topic selection	7 days	Thu 8/2/07	Fri 8/10/07
3	Discussion about the scope of the project	5 days	Mon 8/13/07	Fri 8/17/07
4	Discussion about various features of system	5 days	Sat 8/18/07	Fri 8/24/07
5	Finalising the problem statement	10 days	Sat 8/25/07	Fri 9/7/07
6	Division of modules	4 days	Sat 9/8/07	Thu 9/13/07
7	Initial Literature Survey	14 days	Fri 9/14/07	Tue 10/2/07
8	First Presentation	1 day	Wed 10/3/07	Wed 10/3/07
9	Extensive Research of Crawlers and Indexers	7 days	Thu 10/4/07	Fri 10/12/07
10	Study of Rank Distribution Algorithm - Page Rank	15 days	Mon 10/15/07	Fri 11/2/07
11	Conglomeration of different existing techniques	5 days	Sat 11/3/07	Thu 11/8/07
12	Designing of a Hybrid model	7 days	Fri 11/9/07	Mon 11/19/07
13	Second Presentation	1 day	Tue 11/20/07	Tue 11/20/07
14	Decision on development platform and backend support	4 days	Wed 11/21/07	Mon 11/26/07
15	Inclusion of special features	6 days	Tue 11/27/07	Tue 12/4/07
16	Final preparation of the design part	4 days	Wed 12/5/07	Mon 12/10/07
17	Third Presentation	1 day	Wed 12/12/07	Wed 12/12/07
18	Implementation of Crawler Module	13 days	Thu 12/13/07	Mon 12/31/07
19	Implementation of fingerprinting	8 days	Tue 1/1/08	Thu 1/10/08
20	Implementation of Compression Decompression technique	15 days	Fri 1/11/08	Thu 1/31/08
21	Implementation of the Parser	10 days	Fri 2/1/08	Thu 2/14/08
22	Implementation of the User Interface	7 days	Fri 2/15/08	Mon 2/25/08
23	Integration of the developed modules	6 days	Tue 2/26/08	Tue 3/4/08
24	Decision on seed links	4 days	Wed 3/5/08	Mon 3/10/08
25	Database connection and population	3 days	Tue 3/11/08	Thu 3/13/08
26	Fourth Presentation	1 day	Fri 3/14/08	Fri 3/14/08
27	Improvement in Parser Technique	7 days	Sat 3/15/08	Tue 3/25/08
28	Implementation of Indexer module	10 days	Wed 3/26/08	Tue 4/8/08
29	Implementation of Refresher module	6 days	Wed 4/9/08	Wed 4/16/08
30	Dividing the application on a distributed environment	12 days	Thu 4/17/08	Thu 5/1/08
31	Project Demo	1 day	Fri 5/2/08	Fri 5/2/08
32	Final Report Documentation	3 days	Sat 5/3/08	Wed 5/7/08
33	Final Report Submission	1 day	Thu 5/8/08	Thu 5/8/08

Fig 16:Gantt chart Schedule

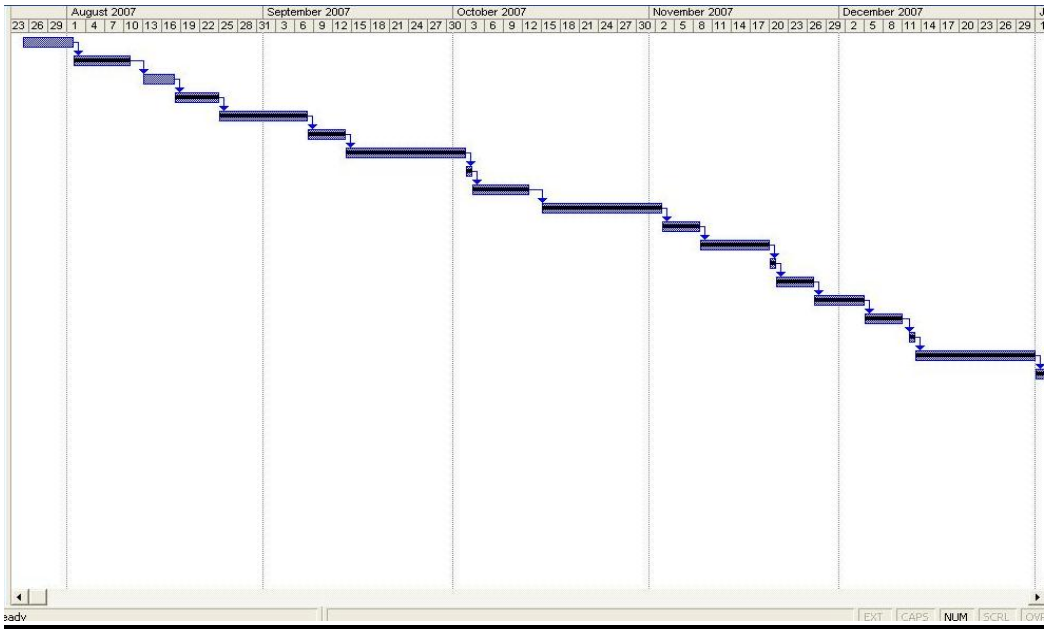


Fig 17:Gantt Chart

(JULY'07-DEC'07)

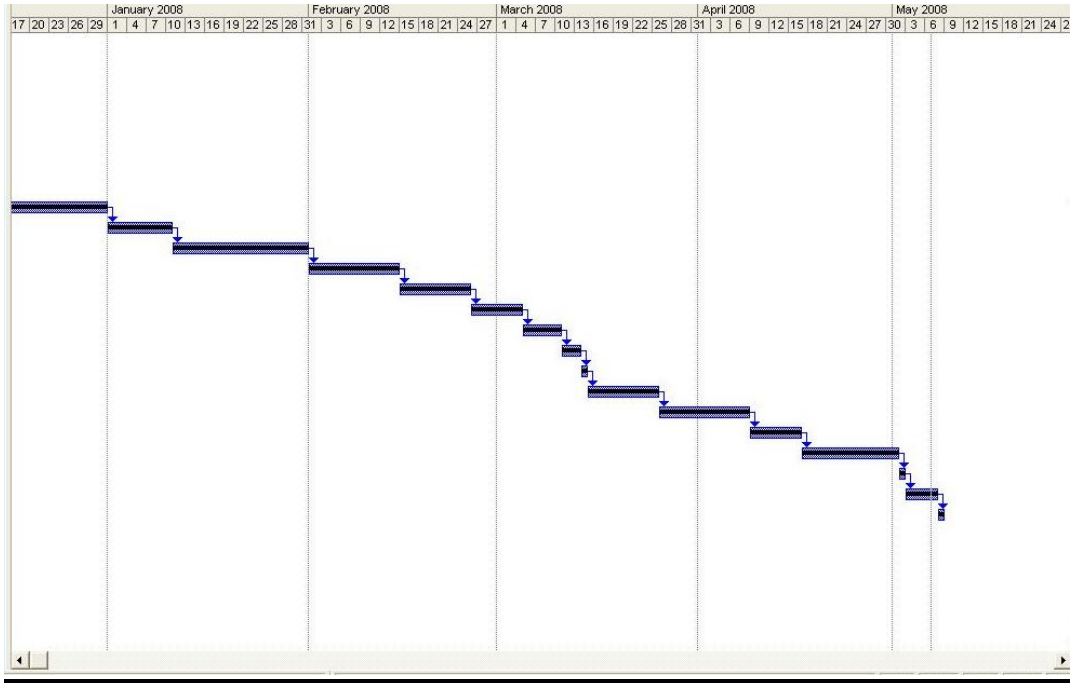


Fig 18:Gantt Chart

(JAN'08-MAY'08)

Appendix F

Annotated Bibliography

S.No	Title	Overview	Conclusion
1.	Dominos: A New Web Crawler's Design Youn`es Hafri, Chabane Djeraba	• In the present paper, the author describes the design and implementation of a realtime distributed system of Web crawling running on a cluster of machines. The system crawls several thousands of pages every second, includes a high-performance fault manager, is platform independent and is able to adapt transparently to a wide range of configurations The paper also provide details of the system architecture and describe the technical choices for very high performance crawling.	• Dominos is a dynamic system, whereby the processes making up its kernel are mobile. 90% of this system was developed using Erlang programming language, which accounts for its highly flexible deployment, maintainability and enhanced fault tolerance. • Despite having different objectives, we have been able to compare it with other documented Web crawling systems
2.	UbiCrawler: A Scalable Fully Distributed Web Crawler Paolo Boldi Bruno, Codenotti,Massimo Santini,Sebastiano Vigna	• The main features of UbiCrawler are platform independence, linear scalability, graceful degradation in the presence of faults, a very effective assignment function (based on consistent hashing) for partitioning the domain to crawl, and more in general the complete decentralization of every task.	• UbiCrawler introduces new ideas in parallel crawling, in particular the use of consistent hashing as a mean to completely decentralize the coordination logic, graceful degradation in the presence of faults, and linear scalability. • The development of UbiCrawler highlighted also some weaknesses

			of the Java API, which we have been able to overcome by using, when necessary, better algorithms.
3.	Scheduling Algorithms for Web Crawling Carlos Castillo, Mauricio Marin, Andrea Rodriguez, Ricardo Baeza-Yates	This article presents a comparative study of strategies for Web crawling. It shows that a combination of breadth- first ordering with the largest sites first is a practical alternative since it is fast, simple to implement, and able to retrieve the best ranked pages at a rate that is closer to the optimal than other alternatives.	For long-term scheduling, the results show that a really simple crawling strategy is good enough for efficiently retrieving a large portion of the Chilean Web. As the idea is to try to keep as many Web sites active as possible, this strategy prioritizes Web sites based on the number of pages available from them, such that is avoids exhausting Web sites too early.
4.	WebCrawler: Finding What People Want. Brian Pinkerton	• This dissertation describes WebCrawler's scientific contributions: a method for choosing a subset of the Web to index; an approach to creating a search service that is easy to use; a new way to rank search results that can generate highly effective results for both naive and expert searchers; and an architecture for the service that has effectively handled a three-order-of-magnitude increase in load. • This dissertation also describes how WebCrawler evolved to	The future of search engines is bright. As the Web continues to expand and increasing numbers of users begin to use it, the role of search tools will become even more important. At the same time, the search engine's job will become more difficult, resulting in many opportunities for research and development. These challenges can be broken up into four main areas: user experience, algorithms for

		accommodate the extraordinary growth of the Web.	high-volume information retrieval, searching more than the Web, and high-volume service architecture.

Publication

(Accepted at "*The International Conference for Artificial Intelligence' 08"*, Las Vegas, USA)

A Hybrid Model for a Search Engine

Shikha Mehta, Ankush Gulati, and Ankit Kalra

Abstract—**The large size and the dynamic nature of the Web highlight the need for continuous support and updating of Web based information retrieval systems such as search engines. Due to resource constraints, search engines usually have difficulties in striking the right balance between time and space limitations. In this paper, we propose a simple yet effective model for a search engine. We suggest a hybrid design which brings together the best features that constitute a search engine. To give an overview, the whole mechanism of how a search engine works is provided. Further, the model is discussed in detail. We then demonstrate how the proposed model, which embodies features like Fingerprinting, Compressor, Importance number and Refresher can improve the efficiency of a simple search engine if applied on a large scale.**

Index Terms—Crawler, Web search engine, refresh policy, search methods, indexing, Distributed information systems.

I. Introduction

THE World Wide Web has grown from a few thousand pages in its early days to more than two billion pages at present. A large number of analyses have been made on the size of the web. Conclusions are drawn that the web is still growing at an exponential pace [2]. Moreover, the web is not structured at all and finding your desired page on the web without a search engine can be a painful task. That is why; search engines have grown into by far the most popular way for navigating the web. In fact, search engines were also known as some of the brightest stars in the Internet frenzy that occurred in the late 1990s.

Engineering a search engine is a challenging task. Search engines rely on massive collections of web pages that are acquired with the help of web crawlers, which traverse the web by following hyperlinks and storing downloaded pages in a large database that is later indexed for efficient execution of user queries. Many researchers have looked at web search technology over the last few years, including crawling strategies, storage, indexing, and ranking techniques as a complex issue. The key to a search engine is, that it needs to be equipped with an intelligent navigation strategy [7], i.e. enabling it to make decisions regarding the choice of subsequent actions to be taken (pages to be downloaded etc). In this paper, we propose an architecture which can be helpful in improving the efficiency of search engines. Our main goal is to improve the quality of web search engines and build an architecture that can search the ever growing web data in a better way.

The rest of the paper is organized as follows. We begin with a definition of a basic search engine in section 2, giving an overview of how does a search engine work and what are its key components. In Section 3, we investigate the need of an advanced search engine. Section 4 suggests some possible solutions for those problems. In Section 5, we the present a 'hybrid' model for search engines- *Swift* which incorporates the best features possible model and discuss every module in detail. Finally, we discuss the scope and future work possible in this direction in Section 6 and outline our conclusion in Section 7.

II. Working of a Search Engine

Internet search engines are special sites on the Web that are designed to help people find information stored on other sites. A search engine basically consists of four parts. Figure 1 shows a basic search engine model [2] :

- *Crawlers*: They search the Internet -- or select pieces of the Internet -- based on important words.
- *Repository*: It stores the complete HTML of every relevant page crawled by the Crawler.
- *Indexer*: It creates an index of the pages they find, on the basis of their linking with other pages.
- *Searcher*: It allows users to look for words or combinations of words in the local repository. The complete working of a search engine is dependent on the flow of data among the above mentioned modules.

A. Crawler

A Web crawler (also known as spider) is a program, which automatically traverses the web by downloading documents and following links from page to page. Crawlers are mainly used by web search engines to gather data for populating the repository. It starts with a few seed pages and then uses the external links within them to attend to other pages. The process repeats with the new pages offering more external links to follow.

We may think that the job of a crawler is over when all the pages have been fetched to the repository once, but there is another important task that a crawler has to perform, refreshing. The Web is not a static collection of pages. It is a dynamic entity evolving and growing every minute. Hence there is a continual need for crawlers to help applications stay current as new pages are added and old ones are deleted,

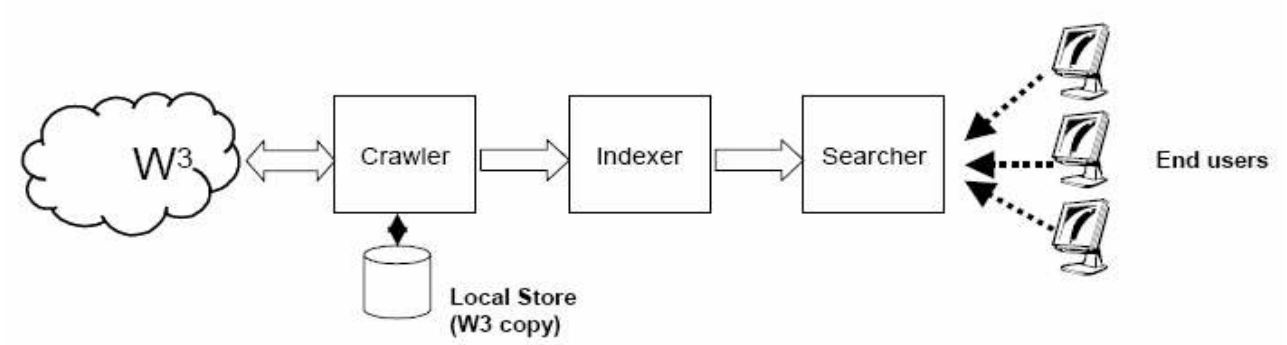


Figure 1 – Model of a Basic Search Engine

moved or modified.

In technical terms, crawling can be viewed as a graph search problem [5]. The Web is seen as a large graph with pages at its nodes and hyperlinks as its edges. A crawler starts at a few of the nodes (seeds) and then follows the edges to reach other nodes. The process of fetching a page and extracting the links within it is analogous to expanding a node in graph search.

B. Local Repository

Everything the spider finds goes into the second part of the search engine, the repository. The repository stores and manages a large collection of 'data objects' in this case web page. All the pages that a crawler crawls and finds relevant are downloaded and stored in the repository. The repository acts as the local cache for this information retrieval system. Whenever, a user searches for a keyword, the searcher module looks into the repository and prints the results.

C. Indexer

An Indexer is a program that "reads" the pages, which are stored in the repository. Even though, each web database has a different indexing method (Brin and Page 1998), the indexer mostly decides what the web site is about and how the website is linked to the rest of the web. It reads the repository, decompresses the documents, and parses out all the links in every web page and stores important information about them to determine where each link points from and to, and the text of the link.

Furthermore, the indexer also does the job of ranking pages on the basis of their importance in the result set. The ranking module consists of a rank distribution algorithm which assigns a random rank to a web page and then computes the rank of other web pages. The algorithms that are commonly used for the purpose of ranking are HITS, Page Rank algorithm and many more [14].

D. Searcher

This is the program that sifts through the millions of pages recorded in the database to find matches to a specific search.

The searcher works on the output files from the indexer. It accepts user queries, runs them over the index, and returns computed search results to the issuer. The searcher is run by a web server and uses the Page Ranks to answer queries.

III. NEED OF AN ADVANCED SEARCH ENGINE

The enormous growth in information that we want to publish on the web has created the need and space for more advanced information retrieval systems to help fetch the information effectively. Many reasons can be cited for the need of an advanced search engine.

- **Complex Structure of the web:** The internet has been aptly named as the Web because of its structure. The web is not organized and its complicated structure creates a lot of problems in effective management of data on the web. The hypertext documents are linked with each other through hyperlinks within them. This gives the user, the ability to choose what he will see next. Interestingly, there can be various links on different pages leading to the same document. A simple crawl mechanism may lead us to a voluminous database with a high degree of redundant data. Thus the crawler needs to have a good crawling strategy [10], i.e., a strategy for deciding whether to download the next page or not, by selecting only one of the many paths available for the same page and hence, avoid data redundancy. It may not seem to be an important issue but when the size and the structure of the web are taken into account, this problem can have deadly consequences on the effectiveness of the search engine.

- **Dynamic nature of the web:** It is an important factor for large-scale Internet search engines. We can broadly classify the issue into three cases :
    1. *Pages Added:* The web is growing in size, and most studies agree that it grows at an exponential rate. This poses a serious challenge of scalability for search engines that aspire to cover a large part of the web. Pages are added everyday and it is the responsibility of the search engine to continuously grow and update its database about the latest link structure of the web.
    2. *Pages Updated:* Apart from the newly

created pages, the existing pages are continuously updated [14]. Newer and more relevant information is added and the older ones are removed. Websites like news portals, etc are updated almost every minute and if the search engine's database is not updated with the current information, it is of no use to the user. Thus, the search engine must store a fresh copy of the pages stored.

3. *Pages Deleted:* Finally, the problem of unavailability of pages also needs to be addressed. The less relevant pages are removed from time to time by the servers. The search engine must keep a check on the availability of the pages it has stored in its local database and remove there links if they are no more hosted.

- **Vast Ocean of data (WWW) to be used as the database to search from:** Size of web cannot be calculated in less than petabytes and crawling the entire web can be a cumbersome task. According to a study, interestingly, the highly relevant content is found very deep in the web. Hence, it can be seen as a problem where we have the limitation of both space and time. In the context of space, we need a local database of size that can store a copy of almost each and every page crawled by the crawler. Taking the time restriction into account, we need to have an efficient algorithm which can search the giant database fast enough to give the desired results in ample time.

IV. Proposed Solution

On the basis of our study of the problems a basic search engine faces, we have come up with a solution which addresses most of the issues discussed in the previous section.

We explored that crawlers consume the maximum amount of resources [9]: network bandwidth to download pages, CPU to evaluate and select URLs, and disk storage to store the text and links of fetched pages as well as other persistent data. Hence, there is a need to improve the working of the crawler considerably.

Issues like **complex structure of the web** can be resolved by using special techniques such as *URL matching* and *Content matching* [10] where in all the pages downloaded by the crawler are first inspected for there content and compared with the copies available in the local database to avoid redundancy.

Similarly, the **large amount of data** can be handled efficiently if it is classified on the basis of some parameters. For example, we can divide the entire database on the basis of content and store pages related to one category in one database and other category in another and so on. We can further reduce the amount of space required to store the data by applying some common compression- decompression techniques [9] on the database.

Also, the **time constraint** can be handled with good *indexing techniques* and we can provide quality search results using a *rank distribution algorithm* [14].

Lastly, there is the issue of **dynamic nature of the web**. This problem can not be easily sorted out. Some smart and effective methods are needed if this issue has to be dealt with. Continuous changes in the web have to be matched with powerful **refreshing techniques** [2]. The local database should be consistently updated with the latest copies of updated pages.

V. Design Of The Proposed Model

In this section we give a detailed presentation of the design of a 'Hybrid' model. *Swift* is a distributed and scalable architecture which is extensible as well. The entire model has been designed in a way that new modules can be added any time for further improvements. Figure 2 shows the complete design of *Swift*.

The key features which make *Swift* a 'Hybrid' model are Compressor-Decompressor, Fingerprint matcher, Importance value calculator, Ranking Algorithm, Focused Crawlers and Smart Refresher algorithm in a Distributed setup.

In the proposed model, the crawling process starts with a few URLs provided. It generates a repository of hundreds and thousands of pages from them and further, refreshes the local database from time to time. The content matching and database updation mechanisms follow the crawling operation. This complete process continues in the background repeatedly. When the user searches for a keyword, the *Searcher* and the *Indexer* modules use the *repository* in its current state as the database to search from. Figure 2 shows the flow of data among the components of *Swift*.

A. Features

- **Distributed Architecture:**

The design proposed is based on a completely distributed architecture. The distribution of jobs to agents is an important problem, crucially related to the efficiency of the system. Therefore, each task must be performed in a fully distributed fashion, that is, no central coordinator can exist. Every agent interacts with either some agent or the *Repository* for taking the input or giving the processed output. Even the *Refresher* and *Extractors* are further distributed for applying the Focused Crawling approach.

- **Fingerprinting:**

Every time a page is downloaded by an *Extractor*, a 64-bit key is generated by applying MD5 algorithm [10] on the contents of the document. We call this key, a fingerprint of this page. This fingerprint can be used by both the *Refresher* and the *Content Seen Tester*. For each newly collected document, if we verify its fingerprint against the fingerprint of the previously collected documents, we can

certainly reduce the data redundancy problem to a large extent and hence, address the space limitations.

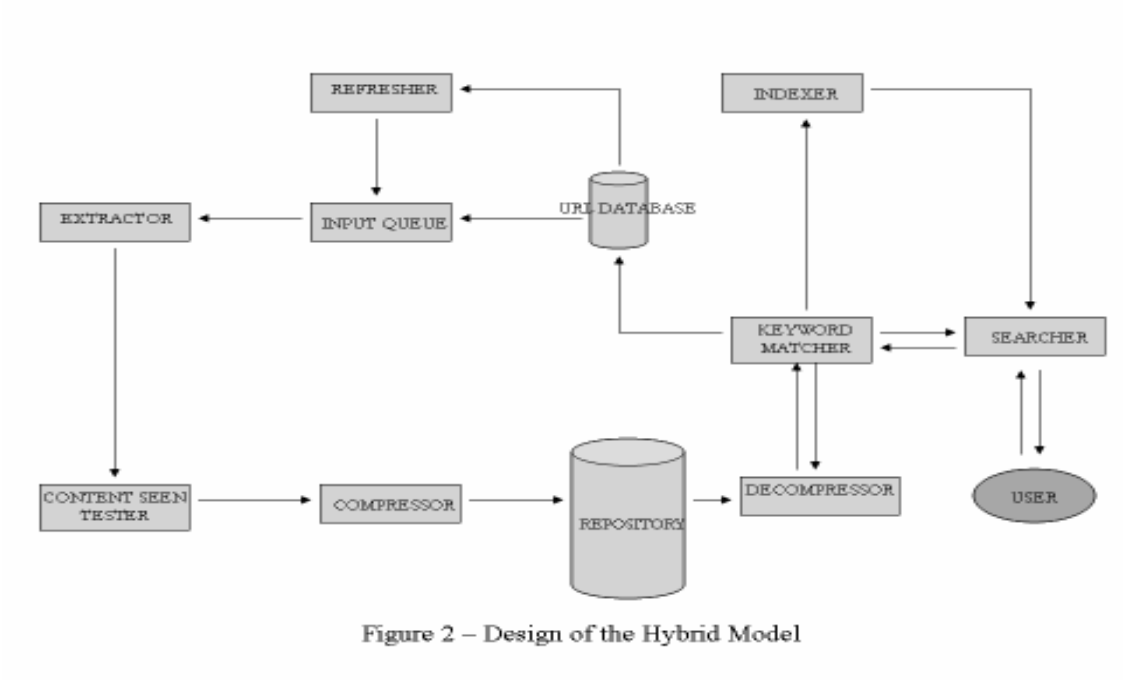


Figure 2 – Design of the Hybrid Model

- **Compression-Decompression:**

We can further reduce the amount of space required to an astonishing degree by using a few simple data compression techniques on the documents before storing them in the *repository*. During the testing phase of this module, we could reduce the size of the local repository by about 60% of the original size. Thus, this feature if taken into account can cause serious improvements.

- **Heterogeneous crawling:**

As the size of the Web grows, it becomes imperative to parallelize a crawling process, in order to finish downloading pages in a reasonable amount of time [13]. This feature suggests a crawler cluster with dedicated machines for crawling the web heterogeneously on the basis of the content.

- **Smart Refreshing techniques:**

Even though there is an established protocol, Robot Exclusion Protocol [7] that can be used to get information about the page. very few websites actually implement this protocol and incorporate it in their pages. We follow the approach suggested by Risvik and Michelsen [2]. This approach uses a relatively simple algorithm for adaptively computing an estimate of the refresh frequency for a given document. Basically, this algorithm decreases the refresh interval if the document was changed between two retrievals and increases it if the document has not changed [2]. This is used as input to the scheduler, which prioritizes between the different documents and decides when to refresh each document. To decide whether the page has changed since the previous crawl, we apply a smart technique. On retrieving the document, we calculate its fingerprint and match it with its existing fingerprint. If they verify, it suggests that page has not changed and thus, we discard the new page. But if they do not verify, the old page is replaced by the new one and its fingerprints are updated in the *URL database*.

- **Importance value:**

A small yet effective feature that gives weight age to the user's choices. Whenever a page is accessed by the user, its *importance value* is raised depicting that the page is a useful one. By default, it is a standard integer value assigned to every URL which gets incremented by 1 every time the URL is visited by the User.

- **Ranking algorithm:**

Due to the Web's size and the fact that users typically only enter one or two keywords, result sets are usually very large. Hence the ranking module has the task of sorting the results such that results near the top are the most likely to be what the user is looking for. In our model, we incorporate the Page Rank algorithm. The crawled portion of the Web is modeled as a graph with nodes and edges. Each node in the graph is a Web page, and a directed edge from node A to node B represents a hypertext link in page A that points to page B [14]. Page Rank is a link analysis algorithm that assigns a numerical weighting to each node of the graph generated with the purpose of "measuring" its relative importance.

B. Working

The working of the model can be explained as follows. To begin with, the *Input Queue* takes a list of seed URLs as its input from the *URL Database* and the *Extractors* repeatedly execute the following steps. Remove a URL from the queue, download the corresponding document, and extract any links

contained in it. With the help of the *Content Seen Tester*, it is ensured that no extracted file is encountered before and there does not exist a copy of the same in the *Repository*. After the document has passed the *Content Seen Test*, its *Fingerprint* is saved in the *URL Database*. The document is then sent to the *Compressor* module which compresses the document and stores it in the *Repository*.

Alongside, the refresher module works on refreshing the populated Repository. It takes the already visited URLs from the URL Database and downloads them. The refresher then applies the refreshing algorithm with the help of Fingerprinting mechanism and updates the local Repository with the fresh copies of existing URLs.

On the other hand, when a user searches for a specific keyword(s), the Searcher module fires a query to the Keyword Matcher. The Keyword Matcher requests the Decompressor to decompress the documents stored in Repository one by one. It searches for the requested keyword(s) within each page it gets from the Decompressor and forwards the results to the Indexer. The Indexer further generates a graph of resulting documents and calculates a rank for each document. The Searcher fetches the results from the Indexer and displays the results in an ascending order of the ranks calculated. Finally, for all the links that are accessed by the User, an update is sent by the Searcher to the URL database to increment its Importance Value.

VI. Future Work

The size of the web is clearly a big challenge, and future evolution of web dynamics raises clear needs for even more intelligent models. One important dimension to be worked upon is the search quality. Search quality means being able to intelligently manipulate the query and fetch results that are as close as possible to the desired output. Features like keyword lexicon can be incorporated in the existing model.

Another important research direction is to study more sophisticated text analysis techniques [8]. At the same time, the "Deep Web" is most likely growing at a rate much higher than the current "indexable" web. There is no unified and practical solution to aggregate the deep web on a large scale.

VII. Conclusion

We have presented Swift, a fully distributed, scalable, incremental and extensible model. We believe that Swift introduces new ideas in intelligent information systems, in particular the search engines. Swift is an ongoing project, and our current goal is to successfully implement the proposed model. We have described the architecture and the operation of Swift in detail. Also, we have discussed the working of a search engine and highlighted how problems arise in all components of a basic search engine model. Swift copes with several of these problems by its key properties like Fingerprinting, Importance Value, Compression-Decompression, Smart Refresher Techniques and Ranking algorithms in a distributed environment. The overall architecture that we have described in this paper is quite simple and does not represent very novel ideas. The system architecture is relatively simple and hence, easy to grow.

References

[1] Qiang Zhu , "An Algorithm OFC For The Focused Web Crawler" in *Proceedings of the Sixth International Conference , Hong Kong, Aug .2007*

[2] Knut Magne Risvik , Rolf Michelsen, *"*Search Engines and Web Dynamics*"*.

[3] Qingzhao , Tan,Prasenjit Mitra , C.Lee Giles, "Designing Clustering-Based Web Crawling Policies for Search Engine Crawlers"

[4] Altigran S. da Silva , Eveline A. Veloso , Paulo B. Golgher "CoBWeb – A Crawler for the Brazilian Web".

[5] Gautam Pant , Padmini Srinivasan , Filippo Menczer " Crawling the Web"

[6] Vladislav Shkapenyuk , Torsten Suel " Design and Implementation of a High-Performance Distributed Web Crawler ".

[7] Younes Hafri , Chabane Djeraba " Dominos : A New Web Crawler's Design ".

[8] Brian Pinkerton " WebCrawler : Finding What People Want ".

[9] Monica Peshave " How Search Engines Work And a Web Crawler Application ".

[10] Allan Heydon < Marc Najork " Mercator : A scalable , Extensible Web Crawler" June 1999.

[11] Junghoo Cho , Hector Garcia-Molina " Parallel Crawlers ".

[12] Carlos Castillo , Mauricio Marin , Andrea Rodriguez " Scheduling Algorithms for Web Crawling ".

[13] Paolo Boldi , Bruno , Massimo Santini , Sebastiano Vigna " UbiCrawler : A scalable Fully Distributed Web Crawler ".

[14] Arvind Arasu , Junghoo Cho , Hector Garcia-Molina , Andreas Paepcke , Sriram Raghavan "Searching the Web".

[15] Behnak Yaltaghian , Mark Chignell " Re-ranking Search Results using Network Analysis - A case study with Google".

[16] Behnak Yaltaghian , Mark Chignell " Effect of Different Network Analysis Strategies on Search Engine Re-Ranking ".

[17] Michelangelo Diligenti , Marco Gori , Marco Maggini " Web Page Scoring Systems for Horizontal and Vertical Search".

[18] Thomas Mandl " Implementation and Evaluation of a Quality Based Search Engine ".

ANKUSH GULATI

SOFTWARE ENGINEER [FRESHER]

Contact Information

C-1/14 Mianwali Nagar, Paschim Vihar, New Delhi. Phone: +91-11-25258214
Email: ankush.gulati@rediffmail.com Mobile: +91-9818049474

Personal Information

Date of Birth 25 Jun 1986
Father's Name Mr. Jagdish Lal Gulati
Gender Male
Nationality Indian

Career Objective

To secure a challenging position that utilizes my education and interests in the fields of technical business and consulting.

Educational Details

Class	Year	School/College	Percentage
B.Tech (Computer Science)	2004-08	Jaypee Institute Of Information Technology University, Noida	CGPA (upto 7th semester) 8.3 (84.84%)
XII CBSE	2004	Hansraj Model School, Punjabi Bagh, New Delhi	77 %
X CBSE	2002	DAV Centenary Public School, Paschim Enclave, New Delhi	84 %

Skills Information

Skill	Details
Key Skills	C / C++ DOTNet (C#, ASP, VB) Java
Familiar With	DBMS HTML, Javascript, PHP Adobe Photoshop Adobe Illustrator

Projects Undertaken

S.No	Details
1.	**SWIFT : A Hybrid Search Engine** **B.Tech Project *[In progress]*** Language : Java, C#, ASP.Net It is a research project that aims at developing a search engine that combines the best features from the existing designs and thus provide a hybrid model for effective search engines. ***A research paper on the proposed design has been accepted by **The 2008 International Conference on Artificial Intelligence** for publication.*
2.	**INDIAFOTOWALA.COM** **Website Development** Language : HTML, JavaScript, PHP ***www.indiafotowala.com*** was a live project. The website was developed for the Ex-Chief Editor of INDIA TODAY, Mr. Dilip Banerjee.
3.	**MOBILE IP SIMULATOR** Language : C# (DOTNet) The project efficiently simulates the Mobile IP Tool using network programming techniques which we used to analyze the characteristics of Mobile IP in real time systems.
4.	**ONLINE COUNSELLING SYSTEM** Framework : **.**NET The aim was to create a simple but efficient online student counseling system for admissions.
5.	**CUSTOMER CARE BACKEND FOR CELLULAR COMPANIES** Language : C/Data Structures This project aims at providing front-end as well as back-end support to customer care executives.

Achievements

1.	Written a research paper on intelligent information systems, "HYRBID MODEL FOR SEARCH ENGINES" accepted by ***The 2008 International Conference on Artificial Intelligence, Las Vegas*** for publication.
2.	Awarded merit prize for performance in 35th "Youth Parliament" Competition by *Ministry of Parliamentary Affairs, Government of India.*
3.	Conferred upon the title of CYBER SCIENCE MASTER on being ranked among the top 1000 students of India's first online Science Olympiad.

Extra Curricular Activities

1.	Contributed in organizing the college fest JIVE'06 as • The Web Developer for the fest. • Coordinated a technical event.
2.	Held the post of "*Literary Activities Minister*" in School Parliament for 2 consecutive years.
3.	Active member of college dramatics society. Won two 2nd prizes at *Le Fiestus '06*, the annual cultural festival at JUIT, Waknaghat in a street play and a stage play competition.