DISTRIBUTED QUERY PROCESSING ON THE DEEP WEB

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

INTEGRATED DUAL DEGREE

(Bachelor of Technology & Master of Technology)

in

COMPUTER SCIENCE AND ENGINEERING

(With Specialization in Information Technology)

By

SUNNY AGGRAWAL

[illegible]

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2012

CANDIDATE'S DECLARATION

I hereby declare that the work being presented in the dissertation work entitled "**Distributed Query Processing on the Deep Web**" towards the partial fulfillment of the requirement for the award of the degree of **Integrated Dual Degree in Computer Science and Engineering (with specialization in Information Technology)** and submitted to the **Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, India** is an authentic record of my own work carried out during the period from May, 2011 to June, 2012 under the guidance and provision of **Dr. Manoj Misra, Professor, Department of Electronics and Computer Engineering, IIT Roorkee**.

I have not submitted the matter embodied in this dissertation work for the award of any other degree and diploma.

Date: June, 2012

Place: Roorkee

[illegible]

(SUNNY AGGRAWAL)

CERTIFICATE

This to certify that the declaration made by the candidate above is correct to the best of my knowledge and belief.

Date: June, 2012

Place: Roorkee

[illegible]

Dr. Manoj Misra
Professor,
E&CE Department
IIT Roorkee, India

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my heartfelt gratitude to my guide and mentor **Dr. Manoj Misra**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his trust in my work, his able guidance, regular source of encouragement and assistance throughout this dissertation work. I would state that the dissertation work would not have been in the present shape without his inspirational support and I consider myself fortunate to have done my dissertation under him.

I also extend my sincere thanks to **Dr. Padam Kumar**, Professor and Head of the Department of Electronics and Computer Engineering for providing facilities for the work.

I would like to thank all my friends who supported and encouraged me to finish this work.

Finally, I would like to say that I am indebted to my parents for everything that they have given to me. I thank them for sacrifices they made so that I could grow up in a learning environment. They have always stood by me in everything I have done, providing constant support, encouragement, and love.

SUNNY AGGRAWAL

ABSTRACT

The WWW is a ubiquitous, open, large heterogeneous, distributed hypertext system. It has become major means of disseminating information. The openness and uncontrolled growth of the web has posed problems in searching or querying the web for relevant information. Currently searching information on the web involves use of search engines which use keywords supplied by the users to search through a large index of static web pages. Most of the webs information is hidden on dynamically generated sites and standard search engines do not find it as they cannot index dynamically generated web pages.

In this dissertation, the design requirements for searching the relevant web documents in dynamically generated web pages are identified. A scheme based on Distributed query processing is then proposed which finds the relevant documents. Then an optimization of a word relevance algorithm is proposed which orders the documents based on their relevance to the query. Simulations for the proposed scheme are conducted for the different user queries. Finally the thesis is concluded by pointing out some open issues and possible direction of future research relating to query processing technique and relevance algorithm.

CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Word Wide Web 1

1.2 Motivation 2

1.3 Problem Statement 3

1.3.1 Problem Description 3

1.4 Organization of Dissertation 3

CHAPTER 2: LITERATURE REVIEW 5

2.1 WebSQL 6

2.1.1 WebSQL system Architecture 6

2.1.2 Query 7

2.2 Discover 8

2.2.1 Discover system Architecture 9

2.2.2 Query 10

2.3 Word relevance Algorithms 11

2.3.1 TFIDF 11

2.3.2 Vector Model 12

2.4 Research Gaps 13

CHAPTER 3: PROPOSED SCHEME FOR SEARCHING ON DEEP WEB 14

3.1 System Architecture 14

3.2 Returning results of queries 16

3.3 Recognizing query completion 16

3.3.1 The CHT protocol 17

3.4 Query Termination 17

3.5 Eliminating Query Recomputations 18

3.5.1 Node-Query Log Table 19

3.6 Optimization of TFIDF algorithm 20

CHAPTER 4: IMPLEMENTATION DETAILS 22

4.1 Design and Development of Basic System 22

4.1.1 Web Query Object 22

4.1.2 User Interface 23

4.1.3 Client Process 23

4.1.4 Server Process 24

4.2 Design and Development of word relevance algorithms 25

4.3 Tools used 25

4.3.1 JavaCC 25

CHAPTER 5: RESULTS AND DISCUSSION 27

5.1 Evaluation of the word relevance Algorithms 27

CHAPTER 6: CONCLUSION AND FUTURE WORK 31

6.1 Suggestions for future work 31

REFERENCES 33

APPENDIX **A1**

A Grammar For HTML Parser A1

LIST OF FIGURES

Figure 1.1 Browsing through hypertext documents 1

Figure 2.1 The WebSQL system 6

Figure 2.2 The Discover user-interface 9

Figure 2.3 The Discover's query server 10

Figure 2.4 TFIDF algorithm 11

Figure 2.5 Vector Model algorithm 12

Figure 3.1 Client process 14

Figure 3.2 Server process 15

Figure 3.3 Multiple visits to server node 18

Figure 3.4 Optimization of TFIDF algorithm 20

Figure 4.1 User Interface 23

Figure 5.1 Relevance of documents to query for dataset 1 28

Figure 5.2 Relevance of documents to query for dataset 2 29

Figure 5.3 Relevance of documents to query for dataset 3 30

CHAPTER 1
INTRODUCTION

1.1 Word Wide Web

The Internet is word wide computer network consisting of millions of computer. The Hyper-Text Transfer Protocol (HTTP) provides medium called Hyper-Media to access information on the Internet. This information is collection of hypertext documents maintained on the computers linked to Internet. A hypertext document allows interaction with it. Along with the information a hypertext documents contains interactive regions called Hyper-Links or Links. A link is reference to another hypertext document. A user views a hypertext document using

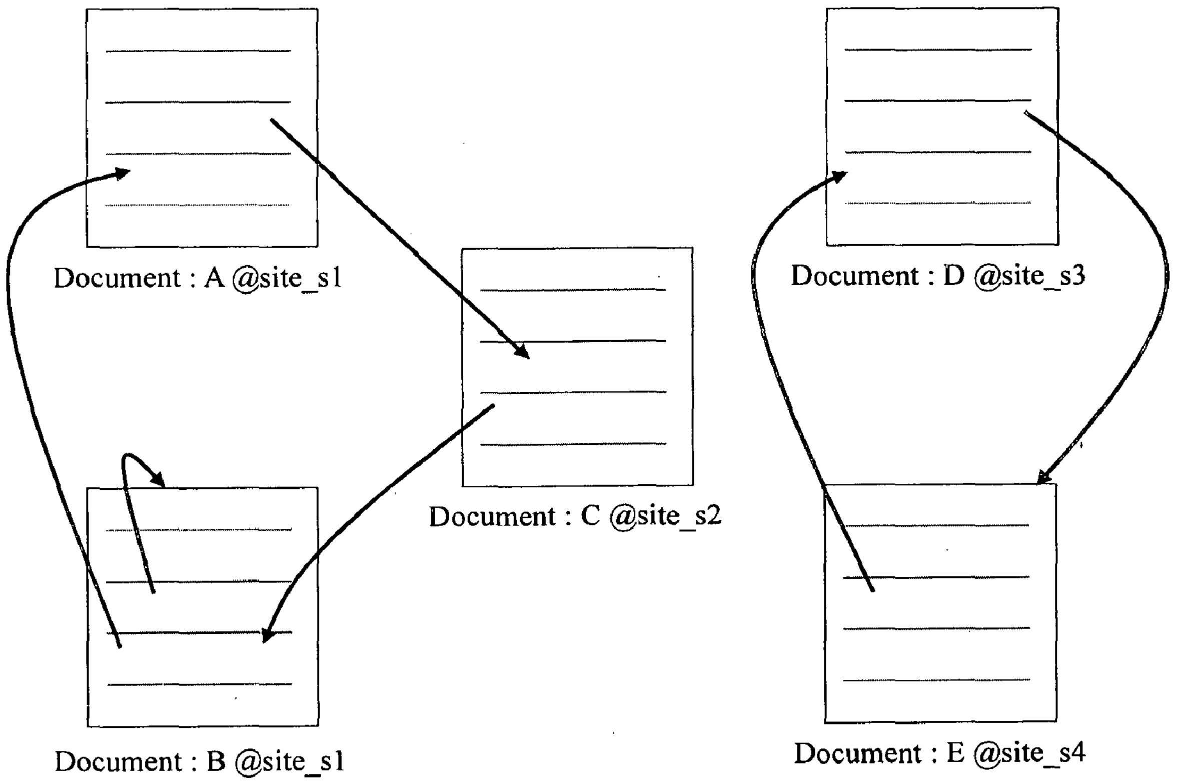

Figure 1.1: Browsing through hypertext documents[6]

a tool called browser. The user interacts with links on this hypertext document using mechanism like clicking the mouse over interactive region. This interaction leads the user to another hypertext document. The user keeps following the links in this manner to browse through subsets of hypertext documents maintained on this hyper-media. In Figure 1.1 hyperlinks contained in hypertext documents residing on different servers is shown. This method of accessing information on the Internet has created an environment coined the word wide web (WWW, Web, W3) [6].

1.2 Motivation

Web can be viewed as enormous database of information. The openness and uncontrolled growth of the web has posed problem of searching the web for relevant information. Currently searching information over the web involves the use of search engines which uses the keywords supplied by the user. Search engines have an indexed database which stores the results for the possible search keywords which is updated time by time. Web crawlers/Robot (a program) is used for making and updating of this large database. When a search engines receives a query then it look for the searched keywords in their indexed database and return the results. Search engines use ranking algorithms for ordering the results. Indexing of webpages works fine as far as the web pages are static HTML (Hyper Text Mark-up Language) pages. When web pages are dynamic (i.e. content of the pages is loaded from database depending upon some parameters) then search engines do not index these pages well. Dynamic pages are a big part of the internet future and it contains more relevant data also so we need a mechanism that works well with both static and dynamic pages.

However search engines do index dynamic pages but their indexing takes much longer time than static web pages. And also number of form parameters is also restricted to only one or two for dynamic sites to get indexed. In this dissertation, problem of searching over the dynamic web pages is addressed. The proposed scheme deals with finding the relevant documents using a distributed query processing technique. Then we use an optimized word relevance algorithm to order the results by their relevance to the query.

1.3 Problem Statement

The aim is to develop and implement an efficient distributed query processor for searching the dynamic web pages which contains the search keywords and order the documents by their relevance order to query.

1.3.1 Problem Description

Search engines use web crawlers for indexing of the web pages. This indexing scheme works well with the static web pages (Surface web) whose content is fixed in the html code. In case of dynamic web pages(Deep Web) where content is loaded from database, indexing of the page may not happen due to limitations of the web crawler[3][4]. So most of the dynamic pages remains hidden and did not appear in the search results. This affects the performance of search engines.

Hence the goal stated in the problem statement can be divided into two sub-problems:

- Search for the web documents that contains the given set of keywords.
- Order the results based on their relevance to the keywords.

1.4 Organization of Dissertation

This report comprises of six chapters including this chapter that introduces the topic and states the problem. The rest of the dissertation report is organized as follows.

Chapter 2 provides a brief description of literature review on the use of distributed query processor in searching the relevant documents and use of search relevance algorithms for ordering the documents.

Chapter 3 provides architecture, design and algorithms description of the proposed scheme for searching the dynamic web pages. Then an optimization of TFIDF word relevance algorithm is discussed.

Chapter 4 gives the brief description of the implementation details i.e. methods, structures and tools used for implementation of the proposed scheme.

Chapter 5 discusses the results and including discussion on them.

Chapter 6 concludes the work and gives the directions for future work.

CHAPTER 2
LITERATURE REVIEW

A distributed query processor searches the web by exploring the hyperlink structure of the WWW. This chapter gives a brief description of the few projects which address solution to the problem in searching the web and then some word relevance algorithms which can be used for ordering of the documents for a given query of words.

For ease of exposition, the following terms are used as define in projects[5][6].

1. **Node**: A node is a web-resource (e.g. HTML documents, XML documents, multimedia files) residing at a web-site.
2. **Link:** A link is the hyperlink from one node to another. It can be of three types: interior, local or global depending on whether the destination of the hyperlink is within the web-resource, on the same server, or on a different server, respectively. For completeness, a null link category which refers to the web-resource itself is also included. In the sequel, we will denote the interior, local, global and null links using the symbols I, L, G and N, respectively.
3. **Web-Query:** This is an alternating sequence of node-queries and PREs, both of which are defined below.
4. **Node-Query:** A query to be evaluated locally on a given node.
5. **Path Regular Expression (PRE)**: Traversal paths on the web are defined using Path Regular expressions, which are built from the above link symbols using the concatenation (.), alternation (|) and repetition (*) operators. For example, N|G.(L*4) is a PRE that represents the set of paths containing the zero length path and all the paths that start with a global link and continue with zero or more local links up-to a maximum of four.
6. **User-site:** The web-server at which the user submits the query.
7. **Start Nodes:** The set of nodes at which the query processing commences.
8. **Query-server:** A Web-server which participates in the distributed processing of the web-query.

2.1 WebSQL

A relational model for the schema less web have been suggested in this project[5]. Each web resource is associated with a tuple in virtual relation:

Document[url, title, text, type, length, modif]

The url is the key and all other attributes can be null. This relation is used for a content query. The structural queries are addressed by accessing each hyperlink in a hypertext document(an HTML file) with a tuple in another relational table.

Anchor[base, href, label]

Where base is the URL of the HTML document containing the link, href is the referred document and label is the link description. The three possible types of hyperlinks in an HTML file recognized are:

- Interior – if the destination is within the source document.
- Local – if destination and source documents differ but located on the same server.
- Global – if the destination and source documents are located on the different servers.

2.1.1 WebSQL system Architecture

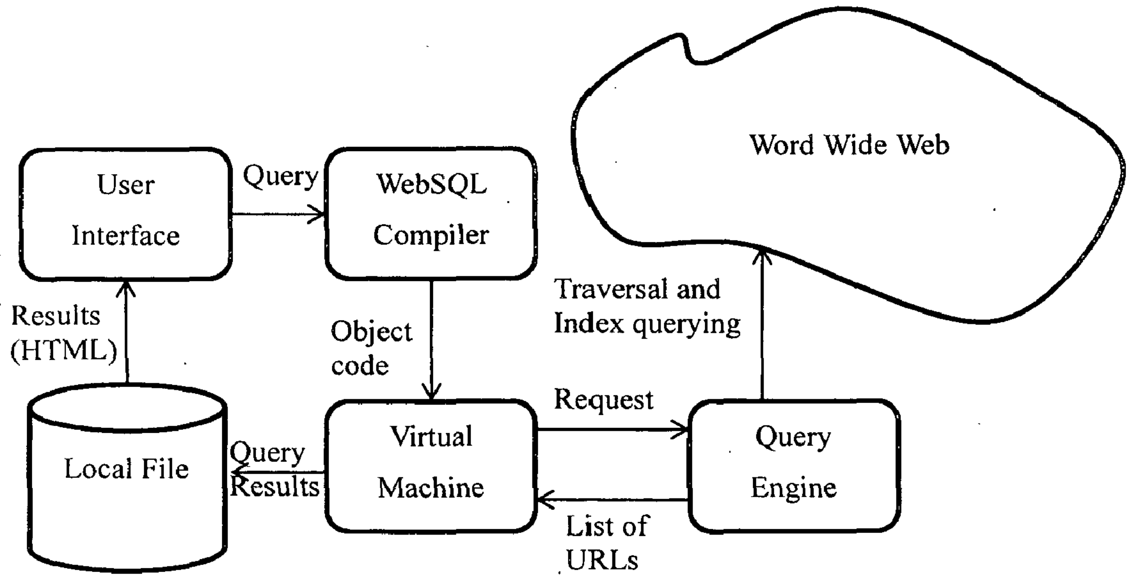

Figure 2.1 The WebSQL system[5]

The WebSQL system architecture is shown in Figure 2.1. The WebSQL compiler parses the query and translates it into a nested loop program in a custom designed object language. The object code is executed by a virtual machine interpreter that implements a stack machine. Virtual machine invokes query engine to perform the actual evaluation, which may involve retrieving the document, parsing it and then building a 'document' and 'anchor' virtual tables.

2.2.2 Query

WebSQL uses Path Regular Expressions (PRE's) to represent path to be traversed on the web for structural query. PRE uses path from the set {I, L, G, N} to form a regular expression, where 'I' denotes interior link, 'L' denotes local link, 'G' denotes global link, 'N' denotes Null link or empty path. For example PRE for a set of paths that start from some node (HTML file) and traverse.

- Two global links followed by at most one interior link; or
- Any number of local links followed by atleast one global link.

is (G.G.(N | I) | (L)*.G.G*)
symbols concatenation (.), alternation(|) and repetitions have their conventional meanings.

Example: Starting from Department of Computer Science home-page, find anchors in all documents such that are linked through paths of length two or less containing only local links. Keep only the documents containing the string 'database' in their title.

WebSQL's Representation:

```
SELECT      a.base, a.href, a.label
FROM        Document d SUCH THAT "http://www.cs.toronto.edu" N | L | L.L d,
            Anchor a SUCH THAT a.base = d
WHERE       d.title CONTAINS "database";
```

Where a is an Anchor table and d is a Document table in virtual relation schemas.

Unlike conventional SQL, here the form clause is evaluated by navigating the web. Documents are downloaded recursively by following the hyperlinks present in the last processed document using the combination of the stack machine and NFA processing over the PRE. On receiving the final document corresponding to the path belonging to the PRE, that document is subjected to where-clause evaluation. If the document satisfies, the temporary virtual database being constructed for this query is updated.

2.2 DISCOVER

This project[6] extends the work proposed in WebSQL but with slightly different interpretation. The Discover system associates the following virtual relations

Document: the discover system maps a node to a tuple in the document relation table. In WebSQL, query processing is local to the user machine and involves multiple downloads of HTML files. So for each node downloaded a tuple is inserted in the Document relation table. Unlike WebSQL in the Discover system query processing is distributed and for a node-query corresponding to this node. This implies that Document relational table will have only one tuple mapped to this node.

Document[url, title, text, textlength]

Anchor: A hyperlink contained in a web resource is mapped to a tuple in this relational table. A user may specify a condition for the hyperlink section.

Anchor[base, label, href, ltype]

Relinfon: This virtual relation provides more flexibility to query information in an HTML file. User may specify an HTML tag as a "delimiter" to be used to split the HTML file into regions called relinfon. Default region is entire HTML file.

Relinfon[delimiter, url, text, textlength]

2.2.1 DISCOVER System Architecture

Discover system can be divided into two parts. First part is the client part as shown in Figure 2.2. User writes a WSQL query using the user-Interface and wait for the results. This query is then parsed by a WSQL parser and converted to a web-query object. This web-query is forwarded to the query server through a socket connection by query dispatcher. Results are collected at the other listening socket at some port.

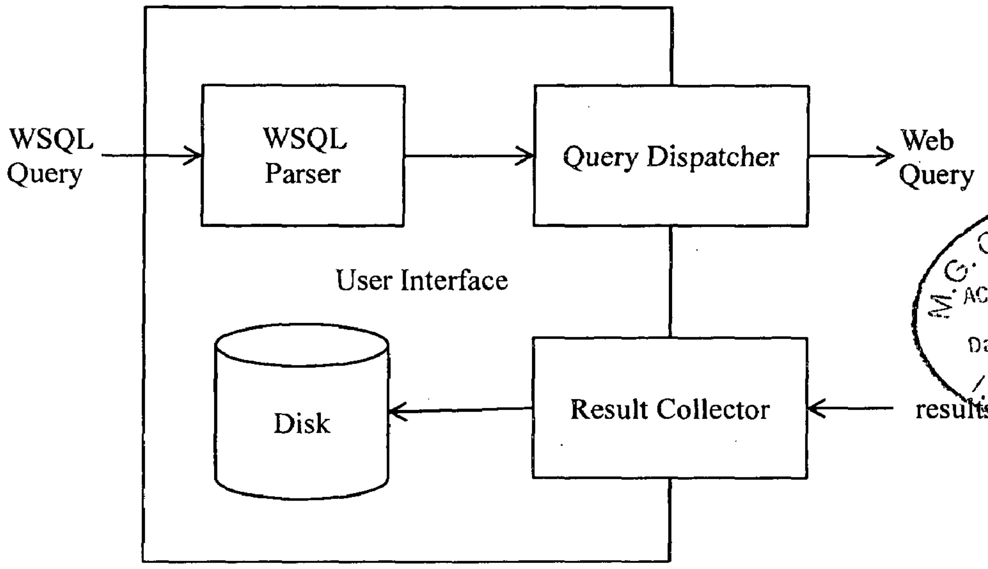

Figure 2.2: The Discover User-Interface [6]

In Discover system each site is assumed to be running a query server, which receives the web-queries and returns the result. Query receiver opens a socket at some fix port to receive the web-queries. It runs as a thread of query server. Pending queue contains the web queries waiting to be processed. It's a thread safe common resource used by query receiver thread and query processor thread. Query Processor then process the queries from the pending queue one by one. For each web-query it access the required HTML file using the url in the query and creates the above mentioned database tables ie: Document, Anchor and relinfon. Query dispatcher is used to forward the query to the next nodes defined in the PRE of the query. PRE parser is used to parse the PRE in the query to obtain the list of next nodes to which the query is to be forwarded. Result Dispatcher returns the results back to the client.

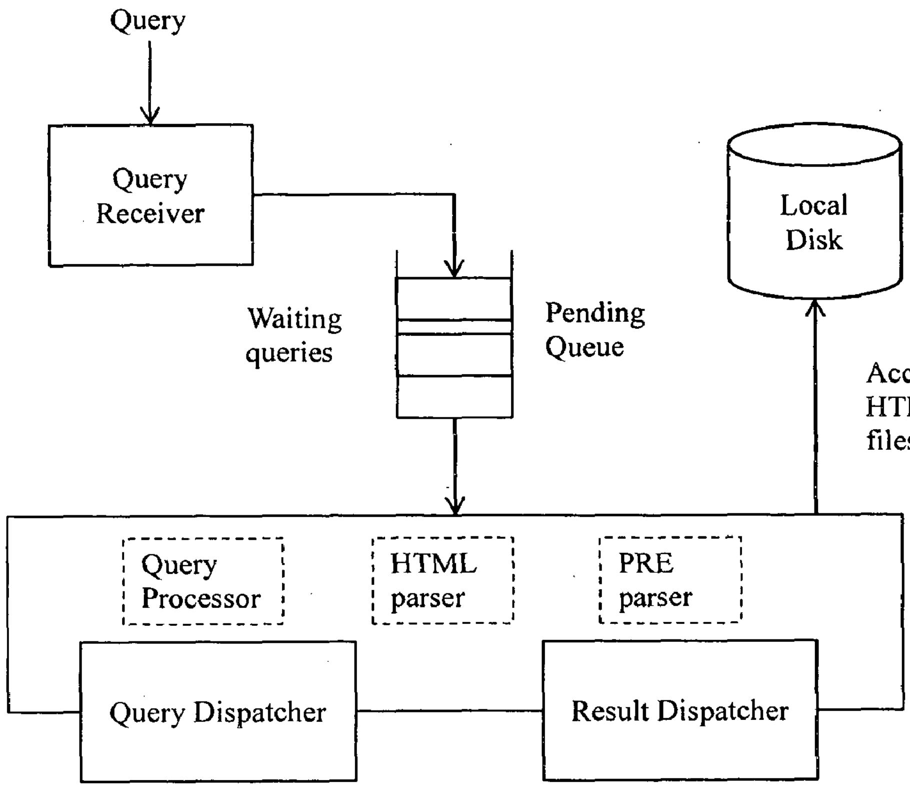

Figure 2.3 The Discover's Query server[6]

2.2.2 Query

Discover also uses PRE's for writing queries, a sample query for the discover system is given below.

Example: Find whether IITR home page contains the word student or not

SELECT d.text

FROM Document d such that http://www.iitr.ac.in L d WHERE d.text CONTAINS "student";

2.3 Word Relevance Algorithms

Word relevance algorithms order the documents by their relevance to the query. Formally We have a set of documents D, with the user entering a query $q = w1,w2, ., wn$ for a sequence of words wi. Then we wish return a subset D^* of D such that for each $d \in D^*$, we maximize the following probability:

$$P(d/q, D) \tag{2.1}$$

2.3.1 TFIDF algorithm

Term Frequency Inverse Document Frequency (TFIDF) algorithm[7] works by determining the relative frequency of the words in a specific document compared to the inverse proportion of that word over the entire document corpus. Intuitively, this calculation determines how relevant a given word is in a particular document. Words that are common in a single or a small group of documents tend to have higher TFIDF numbers than common words such as articles and prepositions.

```
for (each document d in D)
     for (each distinct word w in d) {
          count[w]++;
          $f_{w,d}$ = no of occurrences of w in d;
     }
}
for (each word w in the lexicon) {
     $IDF_{w,D}$ = log(|D|/count[w])
     for (each document d in D) {
          $w_d$ = $f_{w,d}$ * $IDF_{w,D}$
     }
}
```

Figure 2.4: TFIDF algorithm[7]

Given D documents, Relevance w_d of a word w in a document d in D can be calculated as

$$w_d = f_{w,d} * log(|D|/f_{w,D}) \tag{2.2}$$

where, $f_{w,d}$ is term frequency, no of occurances of w in d divided by total no of words in d

|D| is the size of document set D and

$f_{w,D}$ is no of documents in D that contains w

2.3.2 Vector Model

Vector Model[8] is used to measure, either of the similarity of the two documents or of the relevance of a document to the query.

Let W be the total number of distinct words in a document set D. Then each document can be considered as a W dimensional vector, where each word is a different dimension. Component of d in the dimension corresponding to word w is equal to frequency ($f_{w,d}$) of the word in the document. If a word does not appear in a document d then component in that direction is 0. Symbolically let $\vec{D}$ be the vector associated with D and let $\widehat{W}$ be the unit vector associated with set D then

$$\vec{D} = f_{w,d} * \widehat{W} \qquad (2.3)$$

```
for (each document d in D){
    for (each distinct word w in d){
        component of d in direction w = $f_{w,d}$
    }
}
for (each document d in D){
    for (each word w of query){
        $w_d = f_{w,q} / |\vec{D}|$;
    }
}
```

Figure 2.5: Vector Model algorithm[8]

Measurement Rule 1: The similarity of documents D and E is measured by the cosine of the angle between $\vec{D}$ and $\vec{E}$, $\vec{D} . \vec{E} / |D||E|$.

Measurement Rule 2: The relevance of document D to query Q is measured as the similarity of D to Q where Q is viewed as a (very short) document, and similarity is measured as in Measurement Rule 1.

2.4 Research Gaps

The following research gaps were identified after critical literature review.

1. Both WebSQL and Discover use a SQL like query language to specify a user query, so user must be aware of writing SQL queries to use these systems.
2. In Discover, for each query a temporary database is created at the server side. This adds a significant overhead at the server because same web-document may be parsed for different-different queries.
3. There order in which the results to the user are displayed is not specified. Results must be ordered by their relevance to the query.
4. In TFIDF algorithm, if a word is contained in all the documents then second factor of the formula becomes zero hence irrelevant of the different frequencies in the documents. So ordering of the document is not possible.

CHAPTER 3
PROPOSED SCHEME FOR SEARCHING ON DEEP WEB

This chapter gives the complete architecture, design and working of the distributed query processing scheme for searching the relevant documents in the dynamic web and also an optimization for the TFIDF algorithm for ordering the documents by their relevance.

3.1 System Architecture

Client side of the architecture is shown in the Figure 3.1. In our system query is specified using a set of English keywords rather than SQL. Along with a query string user also need to specify a start node(s) and the search space for the query. Start node(s) are the nodes where the search process begins. It is the task of the query dispatcher to forward the query to the start node(s). Path that a query will take is specified using the search space and depth parameters. Search space is either local or global or both where local and global have the same meaning as previously, depth parameters limit the depth of the hyperlink structure to be explored. After the query is dispatched, client starts listening to a socket for the results.

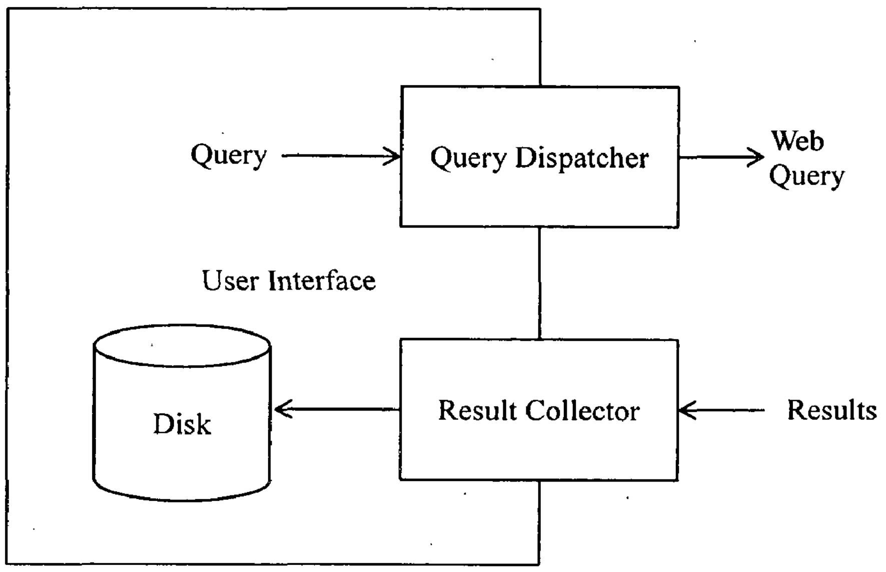

Figure 3.1: Client Process

As in Discover here also we assume that all query servers involved in the distributed processing of these queries are hosting query processors that are capable of receiving and locally processing the query. When a query is received at a query server, query-server completes its local processing of the query and sends back the generated results to the user-site. Further, it forwards the query to another set of query-servers. This set of query-servers is determined from the hyperlinks contained in the searched document. These query-servers also perform similar query processing operations and the process continues until all search space has been fully explored.

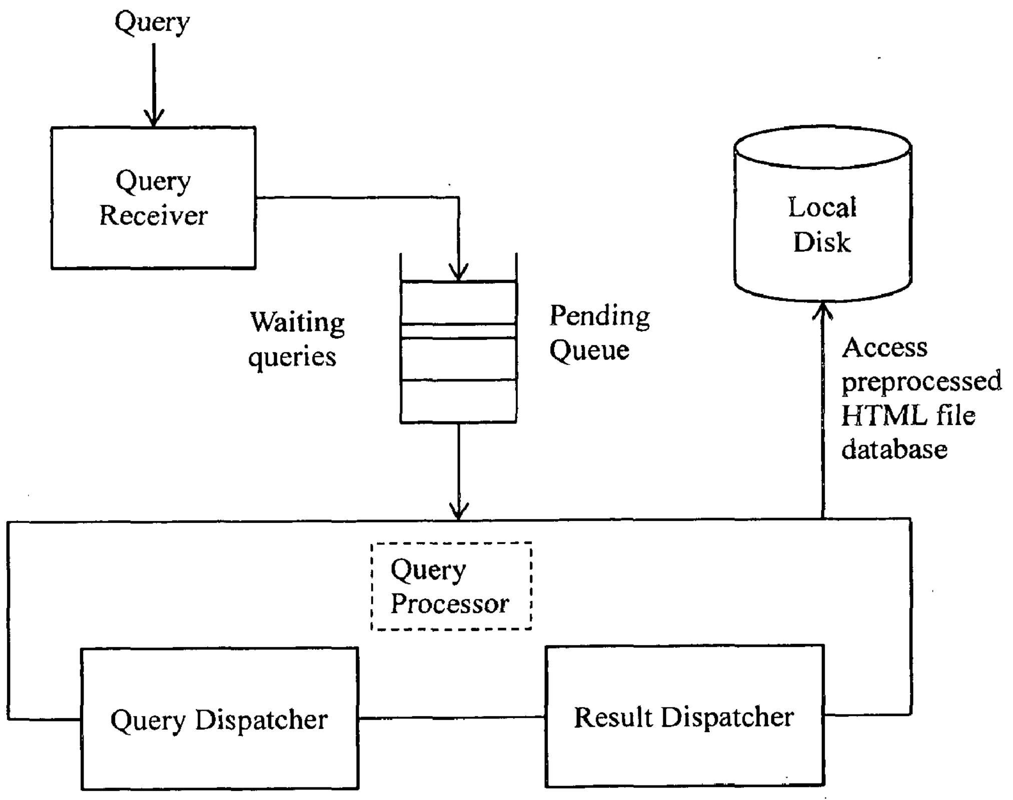

Figure 3.2: Server Process

To process a query, a query-server searches the database. Query server contains a pre-processed word to frequency map for all the web pages that may exist on the server. For the given query, it identifies which page to access and then return the frequency counts of each word in the databse of that web page.

After all the results have been received at the client, we use the word relevance algorithms (which are based upon the frequency count of the words) to order the documents by their relevance to the query.

3.2 Returning results of queries

One option for returning the local results of a query to the user would be to trace back the same path that the query took in reaching this site. However, this option has serious drawbacks: First, the entire history of nodes visited along the path will have to be maintained – that is, we cannot forget the past. This may prove expensive in terms of both storage and bandwidth utilization. Second, if the path is long, the results may take considerable time to reach the user. Finally, it requires query-servers to not only forward queries but also to forward results (in the reverse direction), thereby increasing the load on these servers. In light of the above, we have instead chosen to set up a scheme wherein the results are *directly returned* from the query-server to the user-site. This is achieved by the user-site opening a *listening communication socket* to receive results – the associated port number is sent along with the web-query. When a query-server wishes to communicate results, it utilizes the IP address of the user-site and the port number which came along with the web-query to directly transmit the results to the user. Note that this scheme does not suffer from any of the drawbacks mentioned for the previous option.

3.3 Recognizing query completion

Since, as described above, clones of the web-query migrate from site to site without explicit user intervention, it is not easy to know when a web-query has fully completed its execution and all its results have been received – that is, how do we know for sure whether or not there still remain some clones that are active in the network. Note that solutions such as "timeouts" are difficult to implement in a coherent manner given the considerable heterogeneity in network and site characteristics. They are also unattractive in that a user may have to always wait until the timeout to be sure that the query has finished although it may have actually completed much earlier. To address the above problem, a special mechanism called the CHT (Current Hosts Table) protocol, described in the next subsection is used.

3.3.1 The CHT Protocol

For each user query submitted at a user-site, the client process maintains a table called the Current Hosts Table (CHT). This table keeps track of all the nodes currently hosting clones of the user query. The attributes of the table are

- URL of the node, and
- The state of the query at the node.

As described earlier in this section, after a clone arrives at a query-server and is processed, the query server determines the set of nodes to which the new set of clones should be forwarded. Before forwarding the query to these nodes, the query-server sends the information in the form of a list of rows to be added to the CHT at the user-site, along with the query identifier. It also adds its own URL and the state of the clone it received on top of the list. When the user-site receives this list, it marks the entry in its CHT corresponding to the top-most entry in the list as deleted. When all the entries in the CHT have been marked deleted, it can be concluded that the web-query has been completely processed.

Note that, at the query-server, only after the list is successfully sent, the clone forwarded to the next set of nodes. The reason we process in this particular order is to ensure that the CHT at the user-site will always have complete knowledge about the nodes at which the query is supposed to be currently executing and will therefore always be able to detect query completion. If the opposite order had been used, it is possible that the query may have been forwarded but the CHT not updated due to a transient communication failure between the current site and the user query site. This could lead to the possibility of the user-site wrongly determining that a query has completed when in fact it is still operational in the Web.

3.4 Query Termination

If a user decides to cancel an ongoing query, this message has to be communicated to all the sites that are currently processing the query. One option would be for the user-site to actively send termination messages to all the sites associated with the nodes listed in the Current Host Table. An alternative would be to purge the query locally at the user-site and to close the listening socket associated with the query – subsequently, when any of the sites involved in the processing of this query attempt to contact the site to return the local results, the connection will fail – this is the indication to the query-server to locally terminate the web-query. Note that since we insist that the CHT related information should first be sent to the user-site before forwarding the query to other sites, we do not run into the problem mentioned in the Introduction of termination messages having to chase query messages along the Web

3.5 Eliminating Query Recomputations

Due to the highly interconnected hypertext structure of the web, a web-query may visit a node multiple times following different paths. As shown in the Figure 3.3 node 4 can be visited by 4 different paths. If we do not detect the duplicate cases and blindly compute all queries that are received, not only is it a waste locally but subsequently the same sequence of steps followed by a previous clone will take place. In effect, we will have a "mirror" clone chasing a previously processed clone over the Web. This will also have repercussions at the user site since the same set of results will be received multiple times and these will have to be filtered. In short, permitting duplicate query processing can have serious computation and communication performance implications.

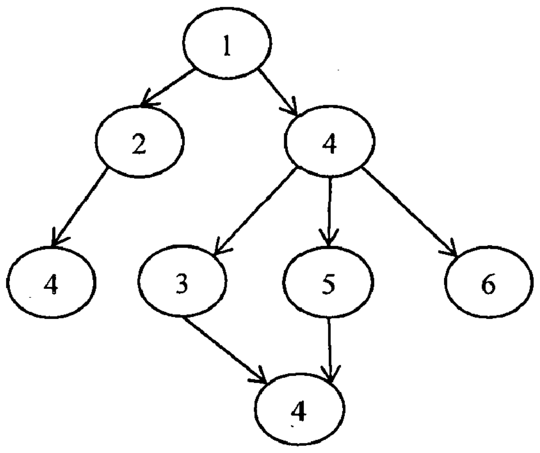

Figure 3.3: Multiple visits to a server node

To deal with duplicate queries, each query server maintains a log table locally described in the next subsection.

3.5.1 Node-Query Log Table

The steps taken by the query server to process the query now involves the following:

1. At each site, we maintain a *log table* which contains the following information with regard to node-queries that have previously visited the site:
    - The global identifier of the query
    - Search Depth

   We denote a log-entry as [Q_{id}, depth].

2. When a new query arrives for a node, check whether an "equivalent" entry exists in the log table. We explain equivalent entries below.

3. If an equivalent entry exists for that node, purge the query for that node, otherwise, make a new entry corresponding to this query in the log table, rewrite the query, if required, (described below) and continue processing the query.

Equivalent Entries in the log table

Whenever a new clone arrives at the site, a new log-table record is created for it and it is checked whether an "equivalent" entry is already present in the log table. Obviously, one kind of equivalence is when the new record is completely identical to an existing entry, that is, they exactly match on all the three fields described above – in this case, the incoming query is dropped.

There are more subtle equivalences, however, that arise when Q_{id} matches but there is difference between the depth parameter, there are two cases to consider. in case where new depth value is lower than the previous value stored in the log table that means that old value is effectively a 'superset' of the new one. Therefor the new entry should not be considered. More complex case is when new value is larger than older one, here some paths have already been considered here but not all. So, here we have a case of the new entry being a "superset" of the existing log entry and we have to ensure that only the difference is processed. Following steps are taken to deal with this case

- Replace the existing log entry with the new entry
- Forward the new query to next set of nodes.

If no equivalence of the above forms can be established with any existing log entry, a new entry is entered into the log table and the query is subsequently processed in the normal manner. To ensure that the log table does not take undue space, the old entries in the table are periodically purged.

3.6 Optimization of TFIDF algorithm

In this section we propose a optimization technique for the TFIDF algorithm. As discussed in the previous chapter idf factor take care of the words like prepositions and articles so that they are not included in the ordering process because they appear in almost all the documents hence idf will approach to zero. But consider the case in which a term other than preposition and article appears in all the documents but with different frequency. Regardless of the different frequencies of the term in the documents, w_d will be zero. To overcome this problem we add an extra factor to the formula which takes into account for such situations.

$$w_d = f_{w,d} * (log(|D|/f_{w,D}) + \log L) \qquad (3.1)$$

where,

$f_{w,d}$ is term frequency, no of occurances of w in d divided by total no of words in d

|D| is the size of document set D and

$f_{w,D}$ is no of documents in D that contains w

L = length of the word

```
for (each document d in D)
     for (each distinct word w in d) {
          count[w]++;
          $f_{w,d}$ = no of occurrences of w in d;
     }
}
for (each word w in the lexicon) {
     $IDF_{w,D}$ = log(|D|/count[w])
     for (each document d in D) {
          $w_d$ = $f_{w,d}$ * ($IDF_{w,D}$ + log(length(w)));
     }
}
```

Figure 3.4: Optimization of TFIDF algorithm

For words with large length, even if they occur in all the documents, their presence in query string cannot be ignored as they can be important words in the search query. So if there are D documents and difference of frequency values is large, in that case document with large frequency count should get preference. In our algorithm the second log term takes care of this. In case all documents have same frequency then this term will just add same constant value to the answer of the original algorithm hence first making this term irrelevant as it appears in all document with same frequency.

CHAPTER 4
IMPLEMENTATION DETAILS

The Distributed query processing scheme discussed in the previous chapter has been completely implemented in Java using JDK 1.6.0[10]. The HTML parser was generated using the *JavaCC*[9] parser generator. The serialization capabilities of Java were exploited to facilitate the forwarding of queries and results from site to site.

4.1 Design and Development of basic system

The four basic components of the distributed query processing system are the Web-Query Object, the User-Interface, the Client Process and the Server Process. In the remainder of this chapter, we describe these components.

4.1.1 Web-Query Object

When a user submits a query, a Web-Query object is formed. Each query is associated with a unique QueryID. To limit the search space of the query another parameter is passed in the query Object. It can have only 3 values: 1 for local only search, 2 for global only search and 3 for both local and global. searchDepth(optional) is the maximum depth of hyperlink structure to be explored, if not provided by the user, 7 is used. queryString is the string to be searched in the web documents. start is the set of start node(s) form where search starts. A port number is also passed in the QueryObject, it is the port at which the client is waiting for the results.

```
struct QueryObject
{
        int Qid;                //Query Id
        int searchSpace;        //1 - local, 2 - global, 3 - both
        int searchDepth;        //depth to be explored
        String queryString;     //search string
        int port                //port at which client is listening for results
};
```

4.1.2 User-Interface

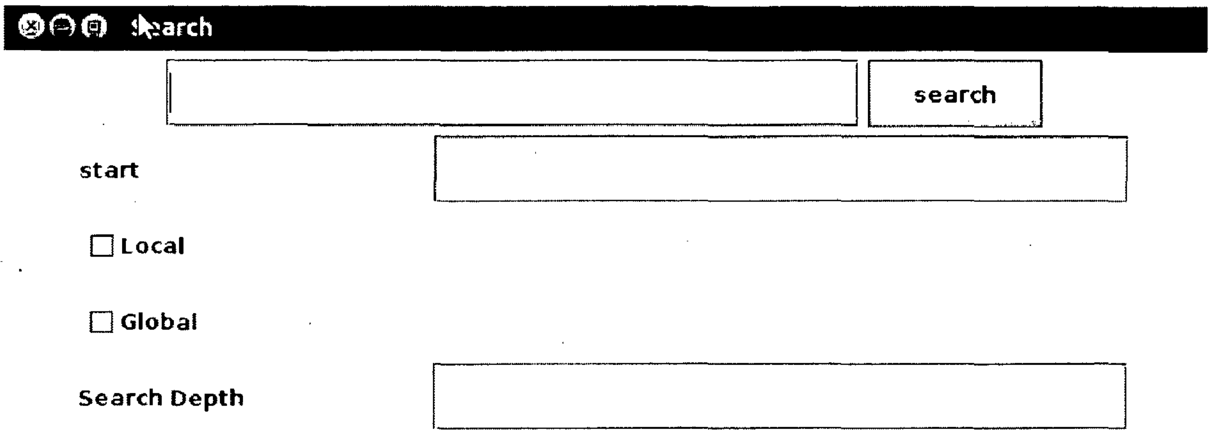

Figure 4.1: User Interface

A simple graphical user interface is provided to the user for searching as shown in figure 4.1, which allows user to write search queries. It forms the web-query object for the user submitted queries and passes the object to query dispatcher.

4.1.3 Client Process

After the user submits the query using the above interface, a series of operations are undertaken by the client process at the user-site.

- The *Result Collector* opens a listening socket at a currently unused port number and waits for results to arrive on this port. The port number is also included in the web-query object.

- The *Query Dispatcher* receives the web-query object along with the set of StartNodes. It makes a socket connection with each of the StartNodes and dispatches the query objects on these connections.

Client program consists of the following methods or sub-modules

void sendQuery (QueryObject Q)

After user submits a query, this function forwards the queryObject to the list of start nodes.

void receiveResults (String result, List CHTnode)

This function is used to collect the results for the query. It opens a listening socket to a choosen port number and starts collecting results until all the entries in the CHT table have been marked.

Here result is a struct of type Results.

```
struct Results
{
  Vector<String> URL;                    //list of urls
  vector<Vector<int> > frequency;        //list of list of freq. received from each node
};
```

4.1.4 Server Process

In our system, Query Server runs as a daemon process at all the web sites. The server is composed of the following components:

1. The *Query Receiver* runs as a thread of the query server and listens on a *common pre-specified port number* at all sites.
2. The *Query Processor* is a thread of the query server and sequentially processes the queue of pending web queries.
3. The *Query Dispatcher* is used by the query processor to forward each web-query to its associated set of NextNodes.

4. The *Result Dispatcher* sends back the results of a query to the user site using the network location information that is encapsulated in the web-query object.
5. The *HTML parser*, which is part of the query processor, builds the pre-processed virtual relation database in memory. A single pass is made over the each possible document at the server and during this process the tuples belonging to the various tables are formed one by one and then inserted into the appropriate table.

void processQuery (QueryObject Q)

This function processes the received query at the server. It computes the result and CHT information to be passed to the client, send it to the client, and forwards the remaining query to the next set of nodes.

4.2 Design and Development of word relevance algorithms

This section gives the implementation details of the algorithms used for ordering of the documents by their relevance. Algorithms discussed in sections 2.3 and 3.5 are implemented in java. All the algorithms take no of documents as a parameter and returns the relevance order with the corresponding relevance values.

4.3 Tools used

JavaCC parser generater tool was used write the parser of html files. Grammar used for the writing the HTML parser is given in APPENDIX A, next section gives the implementation details of the parser using javaCC

4.2.1 JavaCC

JavaCC is a parser generator is a tool for java that reads a grammar specification file and generates an equivalent Java program. Grammar file (filename.jj) starts with optional settings for all the options offered by JavaCC. Following this is a Java compilation unit enclosed between "PARSER_BEGIN(name)" and "PARSER_END(name)".
Following this is a list of productions. Each production defines its left-hand side non-terminal followed by a colon. This is followed by a bunch of declarations and statements within

braces which are generated as common declarations and statements into the generated method. This is then followed by a set of expansions also enclosed within braces.

Lexical tokens (regular expressions) in a JavaCC input grammar are either simple strings or a more complex regular expression. All complex regular expressions are enclosed within angular brackets.

This file is then compiled using javacc and javacc generates a java parser for the defined grammer.

CHAPTER 5
RESULTS AND DISCUSSION

The following sections discuss the performance of the proposed word relevance algorithm for ordering of the web pages.

First two Datasets for test is constructed by querying the Google search engine using two different queries. First 30 results are formatted as the text documents containing the English text only. Third dataset containing 30 documents is constructed using the words chosen from the list of the dictionary word merged with the query words.

5.1 Evaluation of the word relevance Algorithms

In this section we present the result of three different word relevance algorithms for ranking the documents.

The three algorithms are:

1. Term frequency inverse document frequency Algorithm (TFIDF)
2. Vector Model (VM)
3. Proposed optimization of the TFIDF algorithm (OTFIDF)

For each algorithm relevance of the query for each document in the dataset is plotted. In TFIDF and OTFIDF algorithm relevance of the query is the sum of relevance values of each word in the query string obtained using equations 2.2 and 3.1 respectively. In Vector model relevance is calculated using the equation 2.3.

For the Vector Model algorithm value of the relevance lies in the range [0, 1], for TFIDF algorithm it lies in the range [0, log(D)] and for OTFIDF it lies in the range [0,log(LD)]. Where L is the length of the word and D is the dataset size.

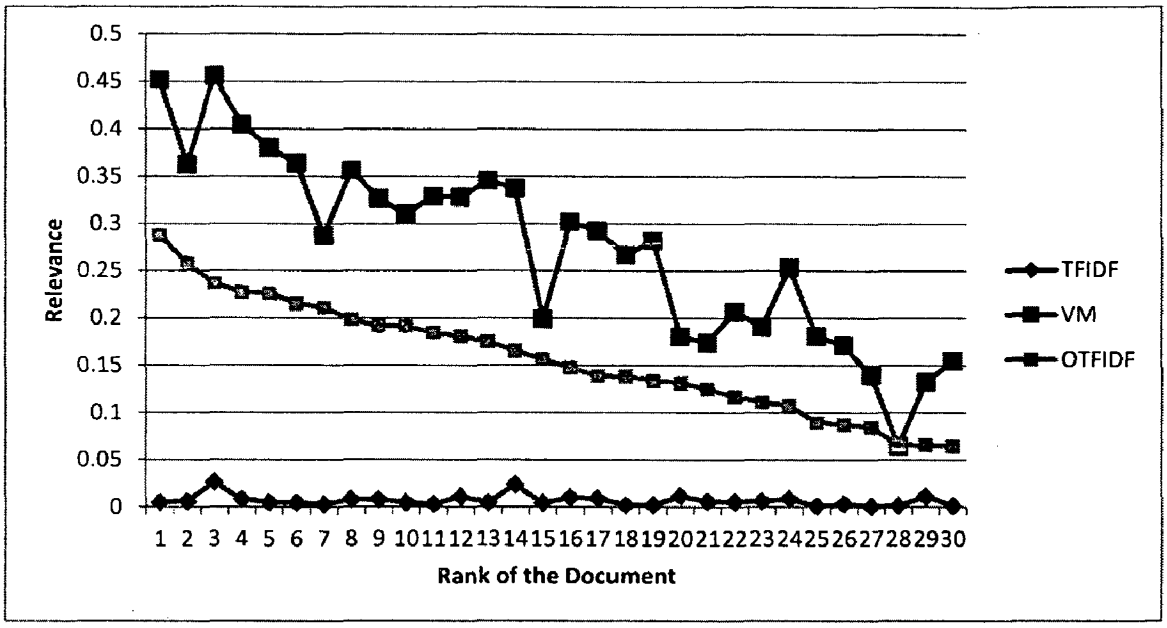

Figure 5.1: Relevance of documents to query for dataset 1

Figure 5.1 shows the relevance values obtained for the first dataset constructed using the search query "difference between process and thread". Relevance values obtained shows that the normal TFIDF algorithm performs poor because most of the words appear in almost all the documents. Vector Model performs better than TFIDF as the relevance values obtained are non-zero. If the query contains prepositions and articles then the relevance values obtained may not reflect the true relevance as these words may be playing big role in the relevance values. OTFIDF algorithm shows improvements of the both issues because for the common words like prepositions and articles both the factors of the formula are very small, and total relevance is sum of the relevance of the other words of the query.

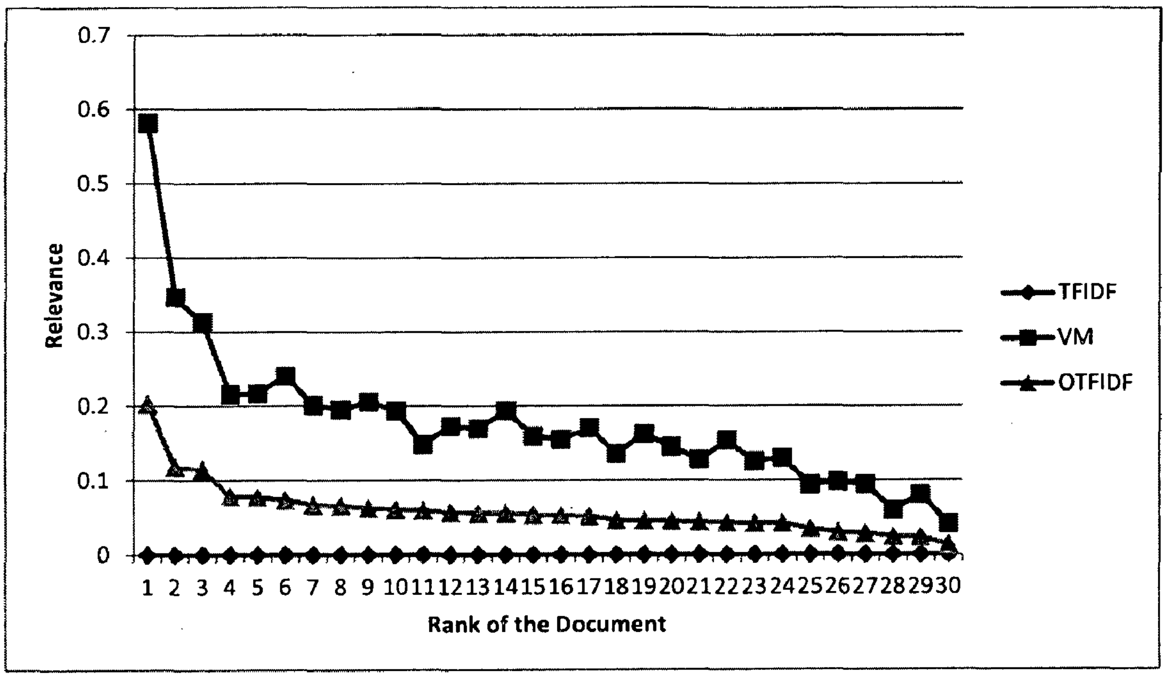

Figure 5.2: Relevance of documents to query for dataset 2

Figure 5.2 shows the relevance values obtained for the second dataset constructed using the search query “Sachin Tendulkar”. First 30 news articles were formatted in the text documents. Relevance values obtained in normal TFIDF algorithm are all zero as all the words appear in all the documents. Again Vector Model performs better than the TFIDF algorithm as the relevance values are non-zero and can be used to order the document. Relevance values obtained in the OTFIDF algorithm performs better than both as it shows gradual decrease in the relevance with the rank of the document.

Figure 5.3 shows the relevance values obtained for the third dataset constructed using the dictionary words merged with the query “dataset”. 30 documents were generated each containing different frequency of the query keyword. Relevance values obtained in normal TFIDF algorithm are all zero as all the words appear in all the documents. Both Vector Model and the OTFIDF approaches rank the documents in the order of the frequency of the query keyword in the document.

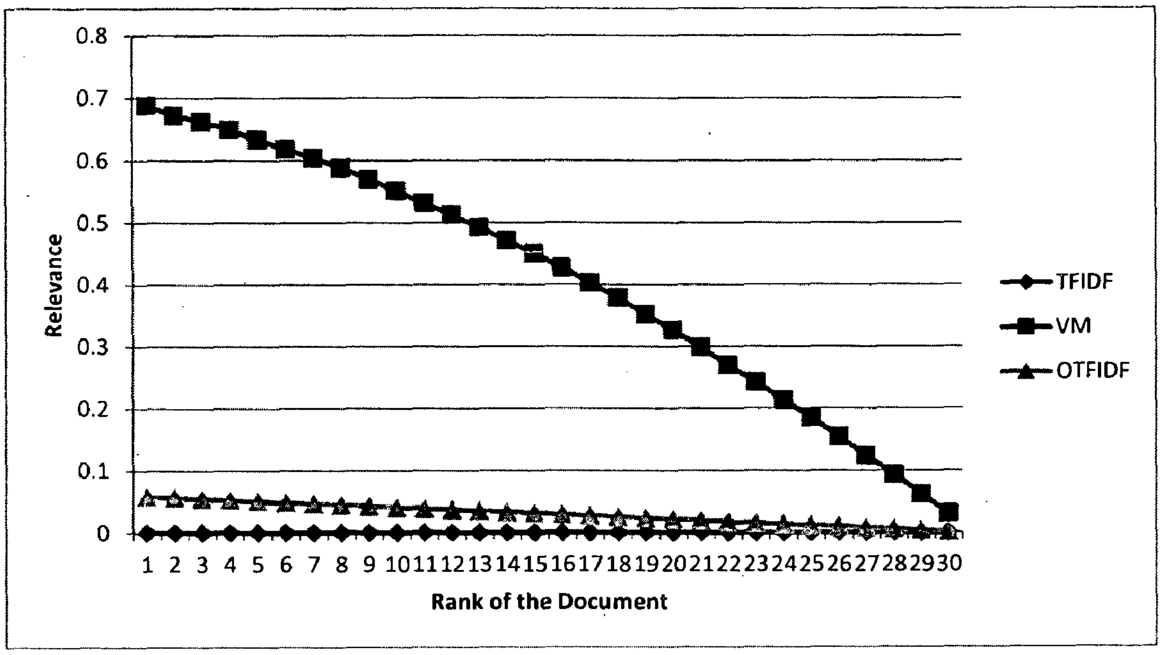

Figure 5.3: Relevance of documents to query for dataset 3

For all the 3 datasets, relevance values obtained for TFIDF algorithm is very low or zero, for Vector Model relevance values obtained are non-zero but it may be influenced by the prepositions, OTFIDF algorithm shows a gradual decrease in the relevance values with the rank of the documents.

CHAPTER 6
CONCLUSION AND FUTURE WORK

The web consists of billions of web documents and searching the web for the most relevant document is the most common need. Currently a large part of the static web pages is indexed by the search engines, but with the growing web and technology more no of sites are moving towards dynamic websites and their numbers are growing with much large rate than static websites. Few studies have been reported on making the dynamic website indexable by putting restrictions on the parameters hence making dynamic website searchable still represents a challenging issue.

In this dissertation, a distributed query processing scheme based approach for searching both the dynamic and static web is presented. The central notion of the scheme is that for each submitted user query, a set of start nodes is defined. Query is initially transferred to these start nodes. This query is forwarded from site to site using the hyperlink structure of the web and results from the each site are directly returned to the client process. When all the results have been received at the client, results are ordered using a relevance algorithm and displayed to the user. The scheme allows users to search the dynamic web pages. Performance comparison of the word relevance algorithms shows that the proposed scheme for ordering the documents is efficient and suitable.

6.1 Suggestions for future work

Since this is an open area for research, the following issues may be addressed in future.

1. The scheme proposed in this dissertation employs preprocessed databases to be created on the server. This may grow very large if no of resulting web pages on the sites is very large. Using some efficient techniques, size of the required memory can be reduced.
2. If some next node is dead, then it may be added to the current CHT at the client, but as this node is dead, no reply will be received and entry corresponding to this node will never get marked in CHT table. This will result in a never ending loop at the client. To avoid this timeout techniques can be implemented that cause the program to terminate.

3. A scheme needs to be implemented for verifying that the server process sending the results are actual results, it is not sending garbage results.

REFERENCES

[1]. Gupta, Nalin and Haritsa, Jayant R and Ramanath, Maya, "Distributed Query Processing on the Web", 16th International Conference on Data Engineering, 2000, pp. 84 - 84.

[2]. Raghavan, Sriram, Garcia-Molina, Hector, "Crawling the hidden web", Proc. 27th International Conference on Very Large Data Bases (VLDB), 2001, pp. 129–138.

[3]. Dynamic page challenges, http://www.feedthebot.com/dynamicpages.html

[4]. Indexing dynamic databased content, http://arnoldit.com/wordpress/2008/04/20/indexing-dynamic-databased-content/

[5]. Alberto O. Mendelzon, George A. Mihaila , Tova Milo, "Querying the Word Wide Web", Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems, 1996, pp. 80-91.

[6]. N. Gupta, "DISCOVER the Web-Database", ME Thesis, Department of Computer Science and Automation, Indian Institute of Science, July 1997.

[7]. Juan Ramos, Juramos Eden, Rutgers Edu, "Using TF-IDF to Determine Word Relevance in Document Queries", 2003.

[8]. G. Salton, A. Wong, C. S. Yang. "A vector space model for automatic indexing". Communication ACM vol. 18, no. 11, Nov 1975, pp. 613-620.

[9]. Java Compiler Compiler, (JavaCC) Version 5.0(Beta), http://www.suntest.com/JavaCCBeta.

[10]. Sun Microsystems. The Java Language : Programming for the Internet, www.java.com

[11]. M. Ramanath, "DIASPORA: A Fully Distributed Web-Query Processing System", Master's Thesis, Indian Institute of Science, 2000.

APPENDIX A
GRAMMAR FOR HTML PARSER

The following grammar was used for the writing html parser. It is used to preprocess the html file into a word frequency map in a single parse of the html file.

- HTMLDocument → (Tag | Decl | CommentTag | <Word> | <Entity> <Punc> | <Space>)*<EOF>
- Tag → (<TagName> (<ArgName> (<ArgEquals> ArgValue)?)* <TagEnd>)
- ArgValue → <ArgName> | (<ArgQuote1> <Quote1Text><CloseQuote1>) | (<ArgQuote2> <Quote2Text><CloseQuote2>)
- Decl → <DeclName> (ArgValue | <ArgEquals>)* <TagEnd>
- CommentTag → <Comment> <CommentText> <CommentEnd>