Crawling the Hidden Web: An Approach to Dynamic Web Indexing

Moumie Soulemane
Department of Computer Science and Engineering
Islamic University of Technology
Board Bazar, Gazipur-1704, Bangladesh

Mohammad Rafiuzzaman
Department of Computer Science and Engineering
Manarat International University
Gulshan-2, Dhaka-1212, Bangladesh

Hasan Mahmud
Department of Computer Science and Engineering
Islamic University of Technology
Board Bazar, Gazipur-1704, Bangladesh

ABSTRACT
The majority of the websites encapsulating online information are dynamic and hence too sophisticated for many traditional search engines to index. With the ever growing quantity of such hidden web pages, this issue continues to raise diverse opinions between the research and practitioner among the web mining communities. Several aspects enriching these dynamic web pages are bringing more challenges day-by-day to index them. By explaining these aspects and challenges, in this paper we have presented a framework for dynamic web indexing. With the implementation of this framework and the results which we have found from it, all the necessary experimental setup and the developmental processes are explained. We have concluded by exposing a possible future scope through the integration of Hadoop-Mapreduce with this framework to update and maintain the index.

General Terms
Web content mining, hidden web indexing, elimination of duplicate URLs, hadoop-Mapreduce for index updating.

Keywords
Dynamic web pages, crawler, hidden web, index, hadoop.

1. INTRODUCTION
In the past few years, there was a rapid expansion of activities in the web content mining area. This is not surprising because of the phenomenal growth of the web contents and significant economic benefits of such mining. Based on the primary kind of data used in the mining process, web mining tasks are categorized into four main types: (1) Web usage mining, (2) Web structure mining, (3) Web user profile mining and (4) Web content mining [7, 14]. Given the enormous size of the web, the indexed web contains at least 13.85 billion pages [9]. Many users today prefer to access web sites through search engines. A number of recent studies have noted that a tremendous amount of content on the web is dynamic. According to [8] Google, the largest search database on the planet, currently has around eight billion web pages which are already indexed. That's a lot of information. But it's nothing compared to what else is out there. Google can only index the visible web, or searchable web which refers to the set of web pages reachable purely by following hypertext links. But the invisible web or deep web [4, 5, 16, 17, 21], "hidden" behind search forms is estimated to be 500 times bigger than the searchable web. However, a little of this tremendous amount of high quality dynamic contents are being crawled or indexed and in particular, most of them are ignored.

The focus of this paper is to provide an automatic indexing mechanism for dynamic web contents which are the part of hidden web. It is same as web content mining as we are extracting the words included in web pages. Here we have tried to come up with a simple approach to crawl the textual portion of dynamic contents with the following methodologies:

- *Dynamic content extraction:*

It means extraction of data, hidden behind the search forms of Web pages, such as search results. Extracting such data allows one to provide services, so that search engines will be benefited by the indexing dynamic contents of web pages as most of the time traditional crawlers avoid them.

- *Form detection:*

Web form with single general input text field is considered. A site like in [13] uses one single generic text box for form submission. Forms with more than one binding inputs will be ignored.

- *Selection of searching keywords:*

Although the Web contains a huge amount of data, not always an optimized search result is generated for a given keyword. Here the method developed for selecting a candidate keyword for submitting a query will try to generate an optimized search result.

- *Detection of duplicate URLs:*

Sometimes two different words may generate same URL twice, which will decrease the efficiency if the same URL is crawled again and again. Detection of duplicate URLs and ignoring them is another try-out of this paper work.

- *Automatic processing:*

There is an automation process for crawling. That is recognizing suitable forms, generating keyword for searching, putting the word in the search bar (using dynamic URL) and making or updating an index for the search results; all of these operations will be fully automatic without any human interaction.

This research work only encompasses dynamism in content, not dynamism in appearance or user interaction. For example, a page with static content, but containing client-side scripts and DHTML tags that dynamically modify the appearance

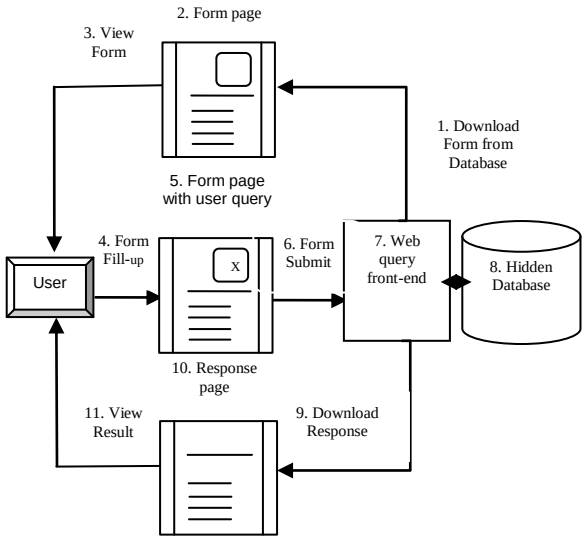

Fig.1: User interactions with search forms to retrieve information from hidden database.

and visibility of objects on the page, does not satisfy our definition as well as our objective.

Outline: Section 2 of this paper contains some aspects of dynamic web pages. Then the proposed approach is presented in Section 3, the framework in Section 4 and its delimitations are discussed in Section 5. Section 6 describes our experimental setup and section 7 shows the analytical parts of our result. The conclusion in Section 8 includes some directions for future work.

2. ASPECTS OF DYNAMIC WEB PAGE

A Dynamic Web Page is a template that displays specific information in response to queries. Its *'dynamism'* lies in its resonance and interactivity, both in client-side scripting and server-side scripting. Dynamic web pages can change every time they are loaded (without anyone having to make those changes) and they can change their content based on what user does, like clicking on some text or an image. All dynamic pages can be identified by the "?" symbol in the URLs, such as

http://www.mysite.com/products.php?id=1&style=a

Visitors find information in a dynamic site by using a search query. That query can either be typed into a search form by the visitor or already be coded into a link on the home page - making the link a pre-defined search of the site's catalog. In that later case, the portion of the link containing the search parameters is called a 'Query String'. This query is then used to retrieve information from the huge database which is hidden behind the search forms.

According to our approach in crawling dynamic web pages, this whole operation is depicted in Fig.1.

2.1 Problems with dynamic page

Most of the search engines avoid crawling dynamic pages for various reasons. Among them, the most common are:

- *Trap:*

A spider trap happens when a search engine's web crawler becomes snared in an infinite circle of a website's coding, which produce endless numbers of documents; web pages that are heavily flooded with characters, which may crash spiders programs**.**

- *Insufficient query information:*

Search engine spiders have a much tougher time with dynamic sites. Some get stuck because they can't supply the information the site needs to generate the page.

- *Security and privacy issues:*

Many of these sites require user authentication and bypassing it automatically may cause violation of privacy or law.

- *Policy preferences:*

Some search engines deliberately avoid extensive indexing of these sites.

- *Costly:*

It needs expertise in contradiction to the static one that is simple and straight forward.

Moreover, Google will not follow links that contain session IDs embedded in them as in [1]. In our approach, we have tried to come up with a simple process to crawl dynamic web pages by bypassing most of these issues.

2.2 Crawling dynamic page

A web crawler is a relatively simple automated program or script that methodically scans or "crawls" through web pages to create an index of the data it's looking for. When a search engine's web crawler visits a web page, it "reads" the visible text, the hyperlinks and the content of the various tags used in the site, such as keyword rich Meta tags. Using the information gathered from the crawler, a search engine will then determine what the site is about and index the information. In practice common crawler algorithms must be extended to address issues like [11]: speediness, politeness, content exclusion, duplicate content detection or continuous crawling.

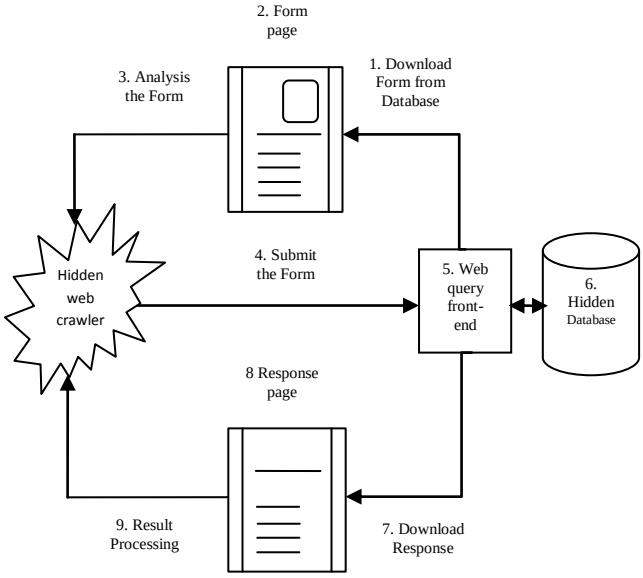

Fig.2: Hidden web crawler.

The crawler designed to crawl hidden web in this paper starts with a list of URLs to visit, called the *seeds.* As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl-frontier. URLs from the frontier are recursively visited according to a set of policies. Web crawling is done on all text contained inside the hypertext content, tags, or text. This operation of hidden web crawler is shown in Fig.2.

2.3 Indexing dynamic page

Indexes are data structures permitting rapid identification of which crawled pages contain like particular words or phrases. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For example, while an index of 10,000 documents can be queried within milliseconds, a sequential scan of every word in 10,000 large documents could take hours. Types of indices data structures include:

A. Inverted index.

B. Forward index

A. Inverted index:

For the process of dynamic web indexing in this paper, we have incorporated an inverted index when evaluating a search query to quickly locate documents containing the words in a query and then rank these documents by relevance. Because the inverted index stores a list of the documents containing each word, the search engine can use direct access to find the documents associated with each word in the query in order to retrieve the matching documents quickly. TABLE 1 is a simplified illustration of an inverted index.

B. Forward index:

The forward index is used to detect duplicate URLs in this paper, because it stores a list of words for each document as simplified in TABLE 2.

According to our proposed approach, to index a set of web documents with the words they contain, we took all documents available for processing in a local repository. Creating the index by accessing the documents directly on the Web is impractical for a number of reasons. Moreover the additional computer storage required to store the index, as well as the considerable increase in the time required for an update to take place, are traded off for the time saved during information retrieval. Collecting "all" web documents are done by browsing the Web systematically and exhaustively and storing all visited pages. This operation is depicted in Fig.3.

TABLE 1: Inverted index of words w.r.t. their URLs

Inverted Index URL	URLs
The	URL 1, URL 3, URL 4, URL 5
Cow	URL 2, URL 3, URL 4
Says	URL 5
moo	URL 7

Table 2: Forward index of words contained in URLs

Forward Index URL	URLs
URL 1	the, cow, says, moo
URL 2	the, cat, and, the, hat
URL 3	the, dish, ran, away, with, the, fork
URL 4	disk, computer, sound

3. PROPOSED APPROACH

In order to index the dynamic contents hidden behind the search forms, in this paper we have come up with an approach which contains the following steps:

3.1 Web pages collection
3.2 Form interface recognition
3.3 Keyword selection
3.4 Query submission
3.5 Result Processing
3.6 Updating the index

3.1 Web pages collection

This part is essentially a static crawling, given the initial URL, the crawler recursively fetches all pages that are linked by it (don't make it recursive, unless you are using functional languages; just use a queue of URLs to be fetched). Test it on the set of web pages created at the beginning. If a page is linked many times, it must be downloaded once. Static crawling is depicted in Fig.4

3.2 Form interface recognition

Recognizing a web form and its fields is a key point of this approach. A standard HTML web form consists of form tags [6], a start tag <form> and an end tag </form> within which the form fields reside. Forms can have several 'input controls', each defined by an <input> tag and some values considered as domain for those input controls. Input controls can be of a number of types, the prominent ones 'text boxes', 'check boxes', 'selection lists' and 'buttons' (submit, reset,

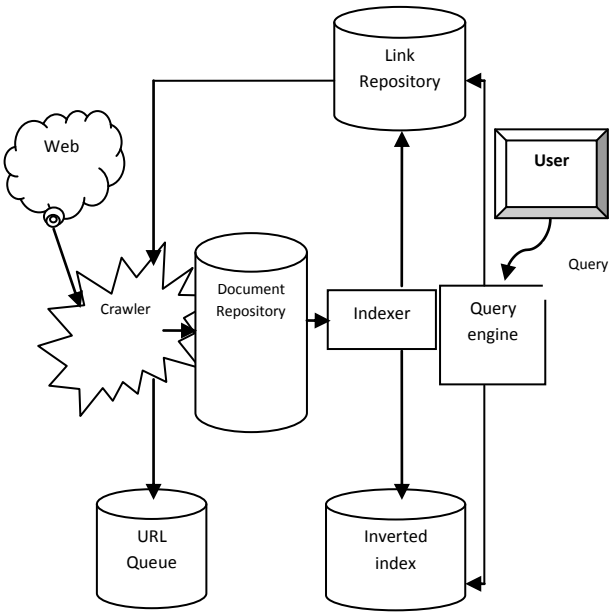

Fig.3: Crawler used in search engines.

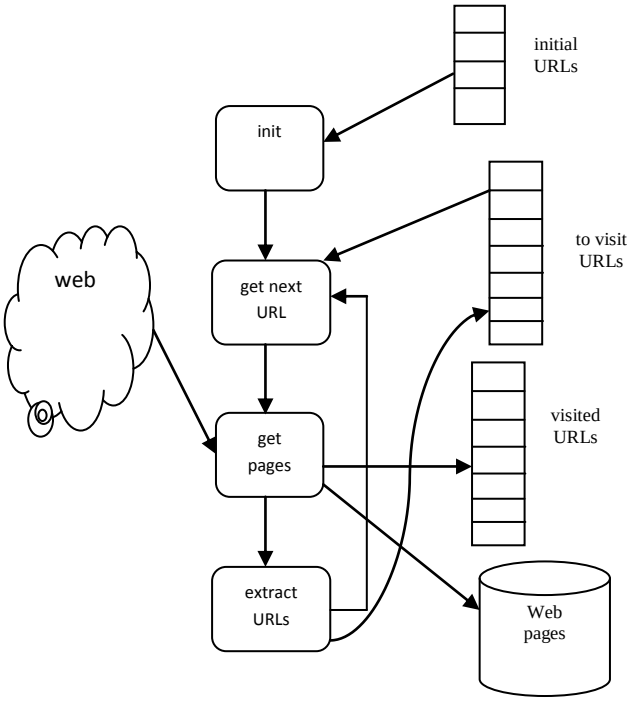

Fig.4: Static Crawling.

normal or radio).Each of the field is having attributes like label, name and values. The form tag also has attributes like 'method' i.e.: 'get' and 'post' and 'action' which identify the server that will perform the query processing in response to the form submission.

In this study, we focus on the forms with one input control binding to a generic text box. Forms with multiple input controls will be ignored. Fig.6 shows such a form with several input controls and Fig.5 shows the piece of HTML markup that was used to generate this form. Whenever our crawler will encounter forms like this they will be discarded from the crawling operation.

But if the crawler encounters a form tag like depicted in Fig.7 it will consider the form as eligible for crawling and will proceed with its operation. A general search form with single input, often on the top-right of the web page is used in this approach.

Further, as per the HTML specification, forms using ***post*** method for form submission are used whenever submission of

```
<form action="MAILTO:someone@example.com" method="post"
enctype="text/plain">
Name:<br /><input type="text" name="name" value="your name"
/><br />
E-mail:<br /><input type="text" name="mail" value="your email"
/><br />
Comment:<br /><input type="text" name="comment" value="your
comment" size="50" />
<br /><br />
<select name="cars">
<option value="volvo">Volvo</option><option
value="saab">Saab</option>
<option value="fiat">Fiat</option><option
value="audi">Audi</option></select>
<br /><br /><br /><br /><br /><br />
<input type="submit" value="Send"><input type="reset"
value="Reset">
</form>
```

Fig.5: HTML form tag markup for sample Form input controls Keyword selection.

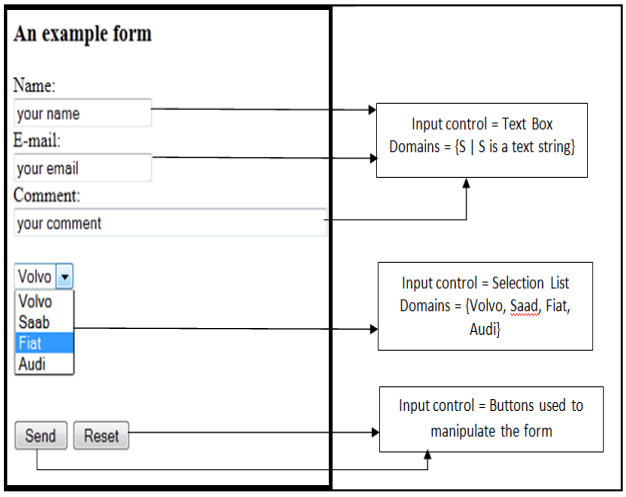

Fig.6: Simple labeled form with several control inputs.

the form results in state changes or side effects (e.g. for shopping carts, travel reservation and login). For these reasons we restrict our attention to those forms which are using ***get*** method to submit the form as they tend to produce contents suitable for indexing.

3.3 Keyword selection

The selection of the most appropriate and relevant value for the input field that can maximize the search is challenging, even though the generic text field generally can accept any keyword. How should a crawler select the queries to issue, given that the goal is to download the maximum number of unique documents from a textual database? Finding the answer for this question is another concern that we have tried to cover here.

In order to solve this question we could select our initial keyword from a dictionary and use it for query submission. But generating a dictionary and searching a keyword within it will be both time and space consuming. Since we want to cover all possible languages, we cannot start from a dictionary of terms. After all an English dictionary will never contain a word which can be used as a keyword for query submission in a Chinese search form. In this case our approach suggests the following aspects:

- *Initial value:*

At the very first, keywords are selected from the static content of the web page having the search form interface.

- *Adaptive:*

After the generation of the 1st set of results, promising keywords are selected from the successfully retrieved pages. Here keywords for query submission in a search form are selected adaptively from itself.

- *Multilingualism:*

By selecting the searching keywords from the web page instead of a predefined location like dictionary or repository our approach also supports multilingualism.

```
<form action=http://www.iutoic-dhaka.edu/dept method="get">
<input name=keyword type=text/>
Input name=searching type=submit value=search/>
</form>
```

Fig.7: HTML form tag markup for a considerable sample form.

- *Limit:*

At most **max** submissions per form will take place to prevent the crawler from falling in a trap (infinite loop). Where *max* is a given number representing the maximum number of queries.

The priority in keyword selection is calculated based on the term frequency F_{tf} in our approach as it determines the importance of a word in a set of documents. The term frequency F_{tf} is a measure of how often a word is found in a collection of documents. Suppose a word 'W_i' occurs 'n_p' times within a web page 'P' and there are total of Np words (including the repeated ones) on that page. Then the term frequency,

$$F_{tf} = n_p / Np. \quad (1)$$

But if we fail to obtain a convenient keyword in that given page, the choice is taken in the repository or at last in the worst case from the dictionary. The selected keywords destined to the query should be compared against the stop words list as these words used to be more frequent.

3.4 Query submission

After selecting the keyword, there is another challenge in submitting the query in the search form automatically, i.e. without any human interaction. Whenever a keyword for a query submission will be selected it'd automatically be submitted in the search form to generate more search results. The action would be something similar depicted in Fig.8.

In this way, whenever a form is submitted a dynamic URL is generated and sent to the database. How many time the query should be submitted and when should it stop? Of course it shall stop when *max* numbers of queries have been submitted.

3.5 Result processing

When our crawler submits a form for processing, different results are possible.

1) The returned page will contain all the data behind the search form.
2) The returned page may contain data, but not showing all the data for the query in a single page. Instead, there may be a "next" button leading to another page of data. In this case, the system will automatically gather all the data on all "next" pages (actually not all, up to a certain limit to avoid a Trap) into a single query result.

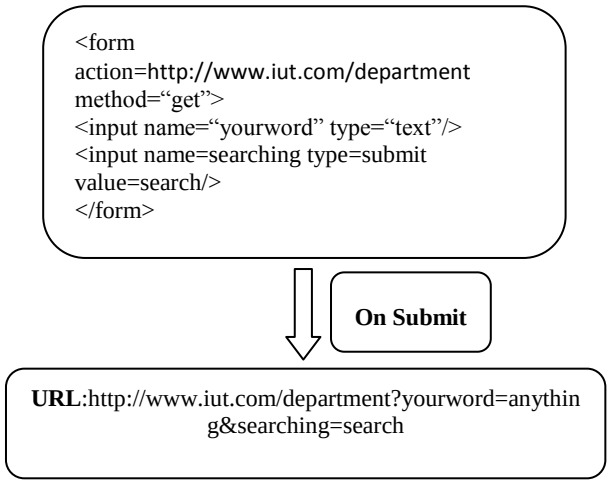

Fig.8: Dynamic URL on query submission.

3) The query might return data, but only part of the data behind the form because the query is just one of many possible combinations of the form fields. In this case the only returned portion will be processed.
4) The query may return a page that not only contains data, but also contains the original form. Here whatever the result is generated we'll gather information as much as possible.
5) The query may return a page that has another different form to fill in. For this case we'll start with the resultant form from the beginning.
6) Some other error cases might involve a server being down, an unexpected failure of a network connection, or some other HTTP errors.
7) The query may go and return the same form requesting for required field to be filled or to be filled with consistent data. Usually this kind of form contains JavaScript.
8) Successive queries may return redundant result, it is therefore important for similarity detection be verified amount successive queries. After all this, the result should be crawled and indexed.

3.6 Updating the index

After the processing the result an initial index will be created. But as this is a continuous process more and more pages will be crawled to extract more words and will be added to the index in times. As a result a continuous updating of the index is required here which will eventually exceed the capacity of a normal single storage device. That's why multiple storage device is needed and in order to do this we have to use "Hadoop-MapReduce" to do the job. Hadoop is an Apache software foundation project as in [20]. It's a scalable, fault-tolerant system for data storage and processing and a framework for running applications on large clusters. Hadoop includes:

- *HDFS* - a distributed file system and
- *Map/Reduce* - offline computing engine.

HDFS splits user data across servers in a cluster. It uses replication to ensure that even multiple node failures will not cause data loss. HDFS breaks incoming files into blocks and stores them redundantly across the cluster. In this approach we are using this splitting and reducing technique to handle the huge amount of index.

4. GENERAL FRAMEWORK

TABLE 3: legend of the framework

Notations	Meaning
Q	Contains crawled words
max	Maximum number of searched keyword=10
W_i	i^{th} word
Limit	Current number of submission
i	Word index
D_URL	Dynamic URL

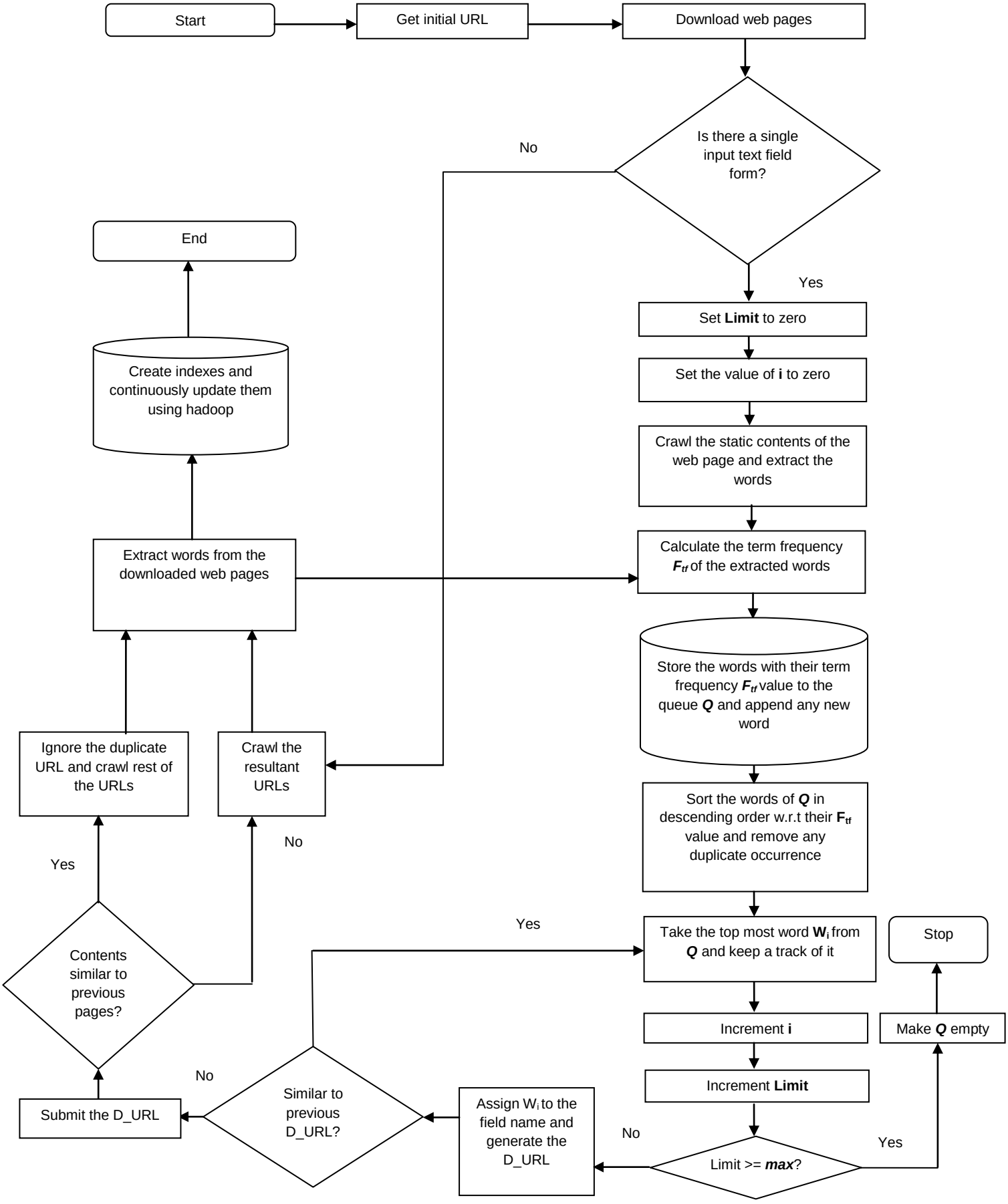

Fig.9: HTML General Framework for dynamic web indexing.

According to Figure 9, the web crawler starts with an initial URL to download the page. After that the downloaded web page is crawled and is checked to see if it contains a single input text field for query submission or not. If not, it will simply be crawled and the words and URLs are indexed. If yes, the limit and variable 'i' are used respectively to count the number of submissions and searching keywords. After that this web page containing the form is crawled and the term frequency Ftf of words extracted from it are calculated and stored in a queue Q. Words in the queue are sorted in descending order of their Ftf and the duplicates are removed. The top most word from Q is assigned to the field name and attached with the submit button name to generate a dynamic URL called D_URL. The D_URL is checked with the previous generated ones for any duplication and if there is a matched, the next top most word will be popped and follow the same process as before. Else if the D_URL is not matched, then it is submitted and related resultant web pages are returned. The jaccard index of the text content of web pages retrieved is calculated to detect for any eventual redundancy, then the duplicates are deleted. Instead of just performing a simple match of D_URLs, duplicates URLs can better be filter as in [18]. At this stages , text are extracted from these new web pages and submitted to hadoop map-reduce were different data structure such from the forward and reverse indexes (re)created in order to update the index. The number of submitted is approximately limited by the number of searched keywords max and when it reaches the limit, the queue Q is emptied and the iteration stops. After this previous whole process the control is transferred the next level of the crawling procedure. More details can be observed from the Fig.9 containing the pictorial representation of the framework. In the Table III serving as the legend of the framework representation, there are some meaning to different notations used. More details can be observed from the Figure 9 and TABLE 3.

5. DELEMITATIONS OF THIS APPROACH

In our approach we are not concerned with the following aspects:

- *Dealing with the form unless it is in the standard format:*

If the code is not properly written in a suitable form, our parser will not be able to conveniently extract information from the web page containing that form. Therefore the presence of the form may not be detected.

- *Handling form that doesn't support passing parameters via URL:*

As in [12] the get method appends its parameters to the action in the URLs in the form of a dynamic URLs format that are often clearly visible (e.g. http://jobs.com/find?src=bd&s=go). In contrast the post method parameters are sent in the body of the HTTP request and its URL is just simple making it difficult for us to deal with it (e.g., http://jobs.com/find).

- *Forms with multiple elements:*

Because we are focusing in only single input form, any form other than this kind will not be considered for submission.

- *Forms that span across several pages:*

This is the case where the same form is extended over multiple continuous pages.

- *Forms with JavaScript embedded:*

Usually input fields of this type of form have a lot of restriction such the type of input, the format, the length, the syntax. Because we are not going to handle all these, we just prefer to ignore them and discard the form.

- *Forms that a single input is not a text field:*

The single input under consideration must be a text field type

- *Forms with personal information:*

Indications such as username, password, E-mail will not be considered for privacy raison.

6. IMPLEMENTATION

6.1 Similarity detection

Determining the degree of similarity on the content of web documents returned from multiple queries can prevent unnecessary redundant from further processing. Using the jaccard index technique we find the similarity of documents and eliminate those that look too similar. The Jaccard index, also known as the Jaccard similarity coefficient (originally coined *coefficient de communauté* by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The formula for calculating Jaccard index $J(d_1,d_2)$ is given below, where d_1 and d_2 denotes two documents as document1 and document2.

$$J(d_1,d_2) = (Td_1 \cap Td_2) / (Td_1 \cup Td_2). \quad (2)$$

J (d1, d2) has some nice properties that are important in the context of a similarity search:

- 1 − J (d1, d2) is a metric called the Jaccard metric.
- For example, the similarity reaches its maximum value (1) if the two documents are identical [i.e., J (d, d) = 1].
- Direct computation of the Jaccard coefficient is straightforward, but with large documents and collections it may lead to a very inefficient similarity search.
- Finding similar documents at query time is impractical because it may raise some computational complexities. Therefore, some optimization techniques are used and most of the similarity computation is done offline.

6.2 Development

The crawler which has been developed in our research work, starts from an initial URL and continuously crawls up to the second level depth. Downloaded web pages after removal of html tags are converted into a set of tokens which are matched against the stopword list and the resulting index includes the forward index, the reverse index and the dictionary. In order to update and maintain our index, using java in a standalone system it is taking considerably a long time to complete. This time complexity can easily be reduced if the updating operation is done using hadoop map-reduce. After having some hard time with the integration of hadoop map-reduce in the framework, we have selected Karmasphere studio coming with a preconfigured hadoop environment in Cloudera as the preferred platform. The implementation of the framework is done in the integrated development environment used by Netbeans version 6.8 with java as programming language with the use of some open source packages such as:

TABLE 4: Result obtained by the crawler in www.dmoz.org

Site	Iteration	Level	Initial	Links	Words R	Words	Time
www.dmoz.org	5	1	97	282	2034	854	
		2		259	3746	1095	0'48
www.dmoz.org	10	1	97	608	4143	1461	
		2		565	7727	1917	1'04
www.dmoz.org	15	1	97	957	6148	2098	
		2		815	11459	2949	1'46
www.dmoz.org	20	1	97	1245	8134	2614	
		2		1171	15552	3533	2'10
www.dmoz.org	25	1	97	1564	10263	3153	
		2		1540	20161	4147	2'58
www.dmoz.org	30	1	97	1872	12379	3685	
		2		1709	23416	4851	3'51
www.dmoz.org	40	1	97	2456	15999	4565	
		2		2294	30145	5889	6'32
www.dmoz.org	50	1	97	3096	20396	5592	
		2		2750	38680	7155	10'51

Here, Site = The name of initial URL, Iteration = Number of unique Queries to be submitted, Level = Level of Query submission, Initial = Number of keywords in the initial URL, Links = How many Links are generated with the given Iteration number, Words R = How many Redundant Words are generated with given Iteration, Words D = How many Distinct (Unique) Words are generated with given Iteration, Time = Time needed to get results up to a certain Level (Here level2)

- httpclient-4.1.1
- httpclient-cache-4.1.1
- httpcore-4.1
- httpmime-4.1.1
- jericho-html-3.2
- commons-codec-1.4
- commons-logging-1.1.1

6.3 Experimental setup

The final experimental from which the above analysis is obtained is done using one personal computer with the following configurations:

- Processor: Intel core2 duo 2.80 GHz
- 4 GB of RAM
- 500 GB hard disk
- 25 to 30 Kbps bandwidth

7. RESULT ANALYSIS

TABLE 4 is an illustration of a sample result from our crawler using [13] as an initial URL. By performing several iterations and keeping track of the number of links, we have come up with distinct words, redundant words, and the execution time at different levels in this table. In general it can easily be remarked that all parameters under observation are increasing satisfactorily as far as the number of iterations is increasing.

Starting with an initial number of 97 words (static crawling) in the first level, in the second level after 5 iterations we are getting around 3746 words (dynamic web crawling). On the other hand, we are getting around 38680 words in the second level after 50 iterations; the same fact is true with links as well. Here the time needed to get the result after a certain level of query submission is totally machine dependant. This bears the example of higher coverage obtained in our approach. An interesting observation to be made here is the advantage of crawling the hidden web over the lonely surfaced web crawling.

Here the time shown is based on a machine whose specification is shown in the "Implementation" chapter. With the increase of machine specifications and the internet speed definitely the amount of execution time will decrease.

8. CONCLUSION

In this research we have studied how to use a hidden web crawler as an approach to dynamic web indexing. We have proposed a complete and automated framework that may be quite effective in practice, leading to an efficient and higher coverage. We have tried to make our design as simple with fewer complexities as possible. Towards the achievement of our goal, we have developed a complete implementation of our proposed framework as in [23] for dynamic web page indexing using java, and the website used as sample input in [13]. Our future work will include a complete implementation, evaluation and analysis of this approach in hadoop, and the derivation of the full algorithm from the framework. We will also try to compare the performance in both java platform and Hadoop MapReduce.

9. REFERENCES

[1] Dan Sisson. *Google SEO secrets, the complete guide*, pp.26–28, 2006.

[2] S. Raghavan, H. Garcia-Molina. Crawling the Hidden Web, in: Proc. of the 27th Int. Conf. on Very Large Databases (VLDB 2001), September 2001.

[3] Dilip Kumar Sharmal, A.k.Sharma2.Analysis of techniques for detection of web search interfaces, *2YMCA University of Science and Technology, Faridabad, Haryana, India*,http://www.csi-india.org/web/csi/studentskorner-december10, *accessed on June,* 2011.

[4] A.Ntoulas, Petros Zerfos, Junghoo Cho, Downloading Textual Hidden Web Content through Keyword Queries, JCDL '05. Proceedings of the 5th ACM/IEEE-CS Joint Conference, 2005.

[5] Luciano Barbosa, Juliano Freire, siphoning hidden-web data through keyword-based interfaces, Journal of Information and Data management, 2010.

[6] http://www.w3schools.com/html/html_forms.asp, *accessed* on, June 2011.

[7] Wiley, Data Mining the Web Uncovering Patterns.(2007) .[0471666556].

[8] Pradeep, Shubha Singh, NewNet- Crawling Deep Web, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, pp. 129-130, May 2010.

[9] http://www.worldwidewebsize.com/, *accessed on June,* 2010.

[10] J Bar-Ilan - Methods for comparing rankings of search engine result-2005, http://www.seojerusalem.com/googles-best-kept-secret/, http://www.search-marketing.info/search-algorithm/index.htm, *accessed on June,* 2010.

[11] David Hawking, Web Search Engines-1, pp. 87-88, 2006.

[12] Jayant Madhavan, David Ko, Luc jaKot, Vignesh Ganapathy, Alex Rasmussen, Alon Halevy. "Google's Deep-Web Crawl", Proceedings of the International Conference on Very Large Databases (VLDB), 2008.

[13] http://www.dmoz.org/, *accessed on June,* 2010.

[14] Brijendra Singh, Hemant Kumar Singh."Web Data Mining Research: A Survey", IEEE, 2010.

[15] http://www.ncbi.nlm.nih.gov/pubmed, *accessed on June,* 2010.

[16] C.H.Chang, M.Kayed, M.R.Girgis, K.F.Shaalan," A survey of web information extraction systems". IEEE Transactions on Knowledge and Data Engineering 18(10), pp.1411–1428, 2006.

[17] P.Wu, J.R.Wen, H.Liu, W.Y.Ma,"Query selection techniques for efficient crawling of structured web sources". In: Proc. of ICDE, 2006.

[18] Wang Hui-chang, Ruan,Shu-hua, Tang,Qi-jie."The Implementation of a Web Crawler URL Filter Algorithm Based on Caching". Second International Workshop on Computer Science and Engineering, IEEE, 2009.

[19] Jeffrey Dean, Sanjay Ghemawat."MapReduce: Simplified Data Processing on Large Clusters". To appear in OSDI, 2004 http://labs.google.com/papers/mapreduce.html.

[20] http://hadoop.apache.org/, *accessed on june,* 2010.

[21] King-Ip Lin, Hui Chen. "Automatic Information Discovery from the "Invisible Web"", Information Technology: Coding and Computing (ITCC'02), IEEE, 2002.

[22] S. Chakrabarti, Mining the web: Discovering knowledge from Hypertext Data, p.67. Morgan Kaufmann Publishers, 2003.

[23] Hasan Mahmud**,** Moumie Soulemane**,** Muhammad Rafiuzzaman**, '**Framework for dynamic indexing from hidden web**',** IJCSI**,** Vol. 8, Issue 5, September 2011.