Effective Web Page Crawler

Dr. Hilal Hadi Saleh* & Dr. Isra'a Tahseen Ali*

Received on: 17/9/2009
Accepted on: 2/9/2010

Abstract

The World Wide Web (WWW) has grown from a few thousand pages in 1993 to more than eight billion pages at present. Due to this explosion in size, web search engines are becoming increasingly important as the primary means of locating relevant information.

This research aims to build a crawler that crawls the most important web pages, a crawling system has been built which consists of three main techniques. The first is Best-First Technique which is used to select the most important page. The second is Distributed Crawling Technique which based on UbiCrawler. It is used to distribute the URLs of the selected web pages to several machines. And the third is Duplicated Pages Detecting Technique by using a proposed document fingerprint algorithm.

Keywords: search engine, web crawl, and fingerprint.

مُجمّع (غواص) صفحات الويب الكفوء

الخلاصة

بسبب تزايد حجم شبكة المعلومات من بضعة الاف صفحة منذ 1993 الى ما يتجاوز 8 بلايين صفحة في وقتنا الحالي, اصبحت محركات بحث الانترنت ذات الاهمية المتزايدة تستخدم كوسائل اساسية في تحديد اماكن المعلومات المطلوبة. ان هذا البحث يهدف الى بناء محرك بحث يعمل على احتواء العدد الحقيقي لصفحات الانترنت اثناء عملية الـ (Crawling) و الفهرسة. لغرض (crawl)الصفحات الاكثر اهمية تم بناء منظومة (crawling)كفؤة التي تستخدم ثلاث تقنيات مقترحه اساسية: الاولى هي تقنية الـ (Best-First)لأختيار الصفحة الاكثر اهمية اولاً, الثانية هي توزيع الصفحات المختاره الى مجموعة من مكائن الـ(crawling) و اللتي بدورها تعتمد على UbiCrawler, و الثالثة تقنية اكتشاف الصفحات المتكررة بإستخدام الخوارزمية المقترحة (بصمة الاصبع النصية).

1. Introduction

Web search services have proliferated in the last years. Users have to deal with different formats for inputting queries, different results presentation formats, and, especially, differences in the quality of retrieved information. Also performance (i.e. search and retrieval time) is a problem that has to be faced while developing such a type of application which may receive thousands of requests at the same time [1].To be more fully understood, the search engine responsibilities, participants are first introduced to the architecture and algorithms of the search engine. With this background, a comprehensible discussion will be

*Computer Sciences Department, University of Technology/Baghdad

https://doi.org/10.30684/etj.29.3.9
2412-0758/University of Technology-Iraq, Baghdad, Iraq

done prior to the work in indexing program, ranking system, and user interface design for search engine server-side and client-side search tools. Figure 1 represents the main parts of a generic search engine that will be explained in detail in the following sections [1].

The search engine's indexer indexes all of its word and phrases and may be the relative position of the words to each other. Later, a user can search this index for the presence of a particular word, phrase or even combination of some words in a web document. Usually, web crawlers store the complementary information for each page, such as time of download and update, different ranks that are computed off-line, header and title, etc [2].

Generic search engines cannot index every page on the Web because the dynamic Web page generators such as automatic calendars, the number of pages is infinite. To provide a useful and cost-effective service, search engines must reject as much low-value automated content as possible. In addition, they can ignore huge volumes of Web-accessible data, such as ocean temperatures and astrophysical observations, without harm to search effectiveness. Finally, Web search engines have no access to restricted content, such as pages on corporate intranets [3].

2. Crawling System:

A crawler, also known as “robot”, “spider”, “worm”, “walker”, and “wanderer” [4, 5], is a program, which retrieves and stores information from the World Wide Web in an automated manner [6]. The first crawler, “Matthew Gray’s Wanderer”, was written in the spring of 1993, roughly coinciding with the first release of NCSA Mosaic [5]. Web crawling is an important research issue. Crawlers are software components, which gather web pages by visiting portions of Web trees, according to certain strategies, and collect retrieved objects in local repositories [2, 6]. Other crawlers may also visit many pages, but may look only for certain types of information (e.g., email addresses), such crawlers are called focused crawlers. At the other end of the spectrum, there are personal crawlers that scan for pages of interest to a particular user, in order to build a fast access cache [7].

A web crawler often has to download thousands of millions of pages in a short period of time and has to constantly monitor and refresh the downloaded pages. As the size of the Web grows, it becomes more difficult or impossible to crawl the entire or significant portion of the Web by a single crawling process [1]. The objective of crawling is to quickly and efficiently gather as many useful web pages as possible, together with the link structure that interconnects them [8].

3. Crawling Strategies:

Sometimes crawls are started from a single well connected page, or a directory such as yahoo.com, but in this case a relatively large portion of the web is never reached. If web pages viewed as nodes in a graph, and hyperlinks as directed edges among these nodes, then crawling becomes a process known in mathematical circles as *graph traversal*. Various strategies for graph traversal differ in their choice of which node among the nodes not yet explored to explore next [9].

3.1 Breadth-First Strategy: In order to build a major search engine or a large repository such as the Internet Archive, high-

performance crawlers start out at a small set of pages (initial URLs or seeds) and then explore other pages by following links in a "breadth first-like" fashion of those pages directly connected with this initial set before following links further away from the start. In reality, the web pages are often not traversed in a strict breadth-first fashion, but by using a variety of policies, e.g., for pruning crawls inside a web site, or for crawling more important pages first [10, 11].

3.2 Depth-First Strategy: The other strategy, Depth-First crawling, employs a narrow, but deep, way of traversing the hypertext structure. This is in contrast to the wide and shallow traversal in Breadth-First approach. Starting from the seed page, the robot picks the first link on the page and follows it, then the first link on the second page, and so on until it cannot go deeper, returning recursively [9].

3.3 URL–Ordering Strategy: This strategy consists of sorting the list of URLs to be visited using some important metrics and crawling the Web according to the established ordering. This technique impacts both the repository refresh time and the resulting index quality since the most important sites are chosen first. Five importance measures are investigated by Garcia–Molina *et al* to establish site importance*: Backlink Count* where the importance is the number of URLs linking to the current URL, *PageRank* which is based on the PageRank ranking metrics, *Forward Link Count*, and *Location Metric* [1, 7, 12].

3.4 Incremental Crawling Strategy: This strategy is concerned with the problem of the data repository freshness. One can choose between two different repository management strategies. The first consists of rebuilding the entire archive from the scratch, and the second consists of updating the changed important pages in the repository and replacing "less–important" pages with new and "more important" pages [1, 13]. The crawler may keep visiting pages after the collection reaches its target size, to *incrementally* update/refresh the local collection. The major difficulty with this approach resides in the estimation of the *freshness* of Web pages needed to reduce the number of *Needless Downloads* [1, 13].

4. Duplicated Web Pages Detecting:

The Internet is the largest public repository of information ever created. Much of this information is published in more than one location. For example, an internet search using the phrase "Linux Documentation Project" results in dozens of almost identical web pages held at different locations, copied from each other and revised slightly. A related issue is that many digital documents are dynamic, continually changing and evolving. It is common practice to keep multiple versions of documents at different stages of development, so it can be necessary to determine whether two documents are different versions of the same text or are different texts altogether. Another problem is plagiarism. Many documents that are published on the Internet are copies or plagiarisms of other documents. Since a plagiarism may not be identical to the original document, using conventional search techniques it can be difficult to distinguish plagiarized documents

from those that are simply on the same topic [14].

There are two types of document fingerprint are full fingerprint and near fingerprint:

a) Full Duplicated Fingerprinting is a technique used in detecting similar documents, rather than using term occurrences and frequency information, fingerprinting aims to produce a compact description, or fingerprint, for each document in the collection. The fingerprint represents the content of the document, and, by comparing these fingerprints, it is possible to determine the likelihood that the documents are co-derivatives [14].

b) Near Duplicated Fingerprint is performed on the keywords extracted from the web documents. First, the crawled web documents are parsed to extracting distinct keywords. Parsing includes removing HTML tags, java scripts, stop words/common words and stemming of remaining words. The extracted keywords ad their counts are stored in the table in a way that the search space is reduced for the detection. The similarity score of the current web document giants a document in the repository is calculated from the keywords of the pages. The documents with similarity score greater than a predefined threshold are considered as near duplicates [15].

A document fingerprint is a collection of integers that represent some key content of the document. Each of these integers is referred to as a *minutia*. Typically a fingerprint is generated by selecting substrings from the text and applying a mathematical function to each selected substring [14].

This function, similar to a hashing function, produces one minutia. The minutiae is then stored in an index for quick access when querying. When a query document is compared to the collection, the fingerprint for the query is generated. For each minutia in the fingerprint, the index is queried, and a list of matching fingerprints is retrieved. The number of minutiae in common between the query fingerprint and each fingerprint in the collection determines the score of the corresponding document [14].

In designing a fingerprinting process, there are four areas that need consideration. The **first** is the function used to generate a minutia from a substring in the document. The **second** is the size of the substrings that are extracted from the document (the granularity). The **third** is the number of minutiae used to build a document fingerprint (the resolution). **Fourth** is the choice of the algorithm used to select substrings from the document (the selection strategy). There have been several methods for fingerprinting, based on variation in –previously mentioned four design parameters [14].

5. The proposed System Architecture:

Engineering a Web search engine offering effective and efficient information retrieval is a challenging task. In particular, the search engine must deal with huge volumes of data. Unless it has unlimited computing resources and unlimited time, one must carefully decide what web pages to retrieve and in what order.

This research aims to enhance the performance of web crawler for web search engines by collecting as

possible as the most important pages, and maximize the download rate. These goals are achieved by implementing an effective –general purpose – Web Page Crawler using multi-threaded distributed crawler that runs simultaneously on as many machines as are available. This distribution is based on UbiCrawler Distributed Crawler. The proposed crawler also respects the robots exclusion protocol and does not traverse pages that are explicitly prohibited from being crawled.

Running a web crawler is a challenging task. There are complex performance and reliability issues. The crawler must carefully decide what URLs to scan and in what order. It must also decide how frequently to revisit pages it has already seen, in order to keep its client informed of changes on the Web. Crawling is the most fragile application since it involves interacting with several web servers and various server names which are all beyond the control of the system. Crawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. A crawler resides on a single machine. It simply sends HTTP requests for documents to other machines on the Internet, just as a Web browser does when the user clicks on links.

5.1 Crawling Algorithm:

In this proposed system, more than 26000 web pages have been downloaded as the main data set used in the proposed Search Engine. The crawler_threads execute simultaneously to fetch contents of the URLs in the urlsToVisit. These threads are also responsible for fetching a page, parsing the page for URLs reachable and partitioning the collected URLs among the different crawler_machines. The algorithm followed by these crawler_threads is:

Algorithm : Crawling Algorithm
Input: S; Set of seed URLs
Output: Collection of Crawled web pages
Step1: Initialization
urlsEncountered = *S*;
urlsToVisit = *S*;
Step2: **while** urlsToVisit is not Empty and thresholdnot greater than 10 **do**
url = get Next urlsToVisit;
robot_exclusion_status = **call** robots_exclusion(url);
If robot_exclusion_status = "not allowed" **then**
goto step 2;
else
threshold= level(url)
page = downloadPage(url);
if content_seen(page) **then**
goto step 2;
else
newUrls =parseForHyperLinks(page);
info=parseFor Info(page);
for all newUrls **do**
if newUrl is Relative **then**
newUrl = make newUrl Absolute;
end if
if urlsEncountered is not contain newUrl **then**
insert newUrl into urlsToVisit;
urlsEncountered insert(newUrl);
end if
end for
call partition_URL_list(newUrls);
end if
end if
end while

The algorithm starts by initializing the set *urlsEncountered* (URLs that are known to the crawler) and the set *urlsToVisit* (URLs that are yet to be crawled) to the seed, S. The seed preferably would be a URL to a web page which would contain lot of hyperlinks. The *urlsToVisit* set is resided in the frontier which is the

data structure that contains all the URLs that remain to be downloaded, as shown in figure 2. *The urlsToVisit* provides two important functions *urlsToVisit.insert(url)* and *urlsToVisit.getNext()* for inserting newly found hyperlinks and obtaining the next URL to crawl, sequentially. Initially it would hold the seed *S* (initial set of URLs), during each step of the crawl one URL is removed from the set using the *getNext()* function. The *getNext()* function is based on the Best_First strategy which refers to the ordering of the URLs based on some priority scheme. The priority scheme is based on using Page Rank value, and the number of hyperlinks coming out/pointing to the page. The data structure used is a priority queue. Now in a repetitive manner one URL from the set *urlsToVisit* is obtained using the *urlsToVisit.getNext()* function.

To make sure that the crawler performs the crawl in a polite manner, robot exclusion status should be checked for each URL before page downloading by calling *robots_exclusion* which returns the status of robot exclusion to get the permission to crawl the web page if it is allowed. Algorithm that returns the status is as the following:

Algorithm : robots_exclusion Algorithm
Input: a URL
Output: Boolean value (true or false) which means "allowed" or "not allowed"
Step1: Look for a "/robots.txt" file on the site.
Step2: **If** found **then**
parse_content("/robots.txt")
for User-agent: *
retrieve Disallow:/string/
{full or partial URL not to visit}
if matches(URL, /string/) **then**
return "not allowed"
else
return "allowed"
end if
end for
else
return "allowed"
end if

Then the level of the obtained URL is returned using *level(url)* procedure which is assigned to *Threshold*. In the proposed crawling system, in addition to *urlsToVisit.Empty* stopping condition, another stopping condition is used, which is called *Crawl & Stop with Threshold*. The *Threshold* represents the number of web pages that are at a depth from the seed page which is equal to 10 in the proposed crawling system.

As it is obvious in figure 2, after obtaining the next URL the server can be contacted and the web page requested for download. To download a page, a connection is open with the http server to obtain a page. Each server would respond to this request in a different manner and speed. A few of these servers could be nonexistent or be very slow to replying. Performing this operation *synchronously* (waiting for completion of one request before placing the next request) could seriously reduce the speed of the crawler. This is overcome by using multiple connections. Managing the connections becomes difficult if one were to allow infinite connections. So, it should be fix the number of connections which obtain individual URLs to crawl from the *urlsToVisit*.

5.2 Mirrored Documents Detection:

There are many cases in which documents are mirrored on multiple servers. Both of these effects will cause any web crawler to download

the same document contents multiple times. To prevent processing a document more than once, a web crawler performs a *Content_seen test* to decide if the document has already been processed. So, in the proposed crawling algorithm, once the web page has been downloaded it will processed by *Content_seen* test to determine whether this document (which is associated with a different URL) has been seen before or not. If so, the document is not processed any further, and the crawler removes the next URL from the frontier.

One of the major difficulties in detecting replicated collections is that many replicas may not be strictly identical to each other. The reasons include:

1. **Update frequency:** The primary copy of a collection may often be updated, while mirror copies are updated only daily, weekly, and monthly. However the mirrors of these collections are usually out of date, depending on how often they are updated.
2. **Different formats:** The documents in a collection may *not* themselves appear as exact replicas in another collection. For instance, one collection may have documents in HTML while another collection may have them in Adobe PDF or Microsoft Word. Similarly, the documents in one collection may have additional buttons, links and images that make them slightly different from other versions of the document.

The content-seen test would be expensive in both space and time if the complete contents of every downloaded document are saved. Instead, a data structure called the *document fingerprint set* that stores a 64-bit of the contents of each downloaded document is maintained.

As shown in figure 2, a document fingerprint is computed for each fetched page. This value is then compared with the fingerprint values of the previously downloaded web pages which stored in the repository, but the fingerprint values are stored in a separated table in SEDB(Search Engine Database).

An improved fingerprint algorithm is used in the proposed search engine. In the following algorithm, all white spaces and special characters are removed to obtain a pure text block, then this lock is partitioned into K-length substrings (K must be efficient as possible as the match is detected). then for each substring a hash value (in the improved algorithm MD5 is used as a hash function) is computed. The number of K-substrings and hence the number of hashes is closed to the size of the document. Simply, it is equal to (m-K+1), where m is the size of the document.

Algorithm: Fingerprint Algorithm
Input: document (a web page)
Output: document's fingerprint
Step1:text=remove_specialchar_whitespace(document);
Step2:list_substring=partition_substring(text,K);
Step3: **for** all substring in list_substring **do**

hashs=hash_function(substring);
end for
step4: list_hashs=subset(hashs,W);
step5: initiate:
right_end=0;
min_index=0;
for all hashs in list_hash **do**
right_end=(right_end +1) **mod** W;
hashs[right_end]=next_hash();
if (min_index=right_end) **then**
i=(right_end-1) mod W;

```
  while not (i=right_end) do
   if (hashs[i]< hashs[min_index]) then
     min_nindex=i;
     i=(i-1+W) mod W
   end if
  end while

   recored(hashs[min_index]);
  else
   if(hashs[right_end]
          ≤hashs[min_index]) then
    min_index=right_end;
    recored(hashs[min_index]);
   end if
  end if
 end for
```

The improvement of fingerprint algorithm will be obvious in the following steps. The set of the hash values are also partitioned into subsets in the same manner that the document is partitioned (each subset has W items). Then, the minimum hash value in each subset is selected. If there is more than one hash with the minimum value, the rightmost occurrence will be selected, but the minimum hash value selected only once. Finally, all selected values are saved as the fingerprints of the document.

As a result of improving the algorithm, it became:

1. ***White-space insensitivity***: In matching text files, matches should be unaffected by such things as extra white-space, capitalization, punctuation, etc. In other domains the notion of what strings should be equal is different—for example, in matching software text it is desirable to make matching insensitive to variable names.
2. ***Noise suppression:*** Discovering short matches, such as the fact that the word *(the)* appears in two different documents, is uninteresting. Any match must be large enough to imply that the material has been copied and is not simply a common word or idiom of the language in which documents are written.
3. ***Position independence:*** Coarse-grained permutation of the contents of a document (e.g., scrambling the order of paragraphs) should not affect the set of discovered matches. Adding to a document should not affect the set of matches in the original portion of the new document. Removing part of a document should not affect the set of matches in the portion that remains.

5.3 URLs Extraction

After the web page is checked, it will be parsed to extract hyperlinks pointing to other web pages. This process requires searching the entire document for HTML <a> and <area> tags and retrieving content of *href* attribute that refers to hyperlinks.

Also, the Crawler extracts important information about the links between two web pages (source and destination) by parsing the source page. The information which is considered as a link attributes will be valuable in computation of the PageRank during the Link-Based

Ranker phase and compute the priority value for each URL which is useful in URL fetching from the frontier. These attributes are:

1. Visibility of the link.

This attribute is determined by checking specific HTML tags which represent the style of the text that used as a link. These two tags are <B>, which means that the text of the link is in **bold** style; and <I>, which means that the text of the link is in *Italic* style. If the hypertext is bold and italic the value of link visibility will be equal to (3). If the hypertext

is either bold or italic the value of link visibility will be equal to (2), other wise it will be equal to (1).

1. Position of the link within the source page.

This attribute is determined by computing the position of the first word in the hyperlink (in other words, the offset of the first word in link's text) within the source page. The source page is partitioned into three parts rather than two parts in order to achieve more accurate results during Link-Based Ranking phase. The value of link's position will be equal to (1) if the link occurs on the least significant one third part of the page, (2) if the link occurred on the middle one third part of the page, (3) if the link is occurs on the most significant one third part of the page.

2. Distance between the source web page and the destination web page.

The distance is found by determining the degree of the differences between two host-names in the URL for (sources and destinations) web pages. But it is now no longer limited to only two cases; the first is if the two host-names are different the value of the distance will be equal to 5, the second case is if both host-names are equal the value will be 1.

In the steps of extracting links, any web crawler will encounter multiple links to the same document. To avoid downloading and processing a document multiple times, a URL-seen test must be performed on each extracted link before adding it to the URL frontier, as it s illustrated in figure 2. The list of URLs that have been crawled is stored in the *urlsEncountered* data structure. It provides two main functions *urlsEncountered.insert(URL)* and *urlsEncountered.contains(URL)* for inserting URLs and checking for duplicate URLs. This can be achieved by searching the set *urlsEncountered* (see Algorithm 1), if a URL is found in *urlsEncountered* then it will be discarded, if not it will be added to the *urlsToVisit* and *urlsEncountered* sets. This is what called URL-seen test; where all the URLs seen by crawler in canonical form are stored in a large table called the *URL table*. The *urlsToVisit.insert()* function would determine the priority of the URL and insert it at the appropriate position in the queue whereas urlsToVisit.getNext() function would remove the first URL from the queue. Again, there are too many entries for them all to fit in memory. Once a hyperlink is found, its URL has to be compared to all the URLs that have been already encountered to avoid duplication.

A typical crawl of the web usually lasts for days or may be weeks. If the system were to crash, say after 20 days of operation then all the data collected till then would be lost and the crawler has to start its crawl from the seed again. Checkpointing is a way of storing the data that has been collected along with the current state of the system onto the disk. Checkpointing could be done 2 or 3 times per day and in the event of a crash the system can be restored back till the most recent checkpoint. Since we have stored the state of the crawler on disk this information can be used to start the crawler from that point onwards. Though checkpointing would reduce the speed of the crawler momentarily it would be of great help in the event of a system failure. The process of crawling is repeated until the set *urlsToVisit* becomes empty or the

crawler stops based on the other condition.

6. System Implementation:

To implement the Crawler require network and text parsing program. This is programmed in Microsoft VB.NET connected to the database in Microsoft SQL Server 7.0. (Note: After the Crawler finishes its work, the Indexer and the Link-Based Ranker could work simultaneously, but in this research they are run sequentially because of the hardware specification which is used as a local server).

On the server side, the process involves creation of a database and its periodic updating done by software called Crawler. The Crawler also called *robot* that store the crawled Web Pages in the repository. The main interface of the crawling process is shown in Figure3.

The first field holds the currently crawled URL which is fetched from the frontier. During the parsing new hyperlinks in the current web page, the URL_id and a full URL are displayed in the interface. Also all extracted information is displayed as a link properties as shown in Figure 3. Then all crawled pages are indexed later.

The front-end of the search engine is the client side having a graphical user interface as in Figure 4, which prompts the user to type in the search query. The interface between the client and server side consists of matching the user query with the entries in the database and retrieving the matched Web Pages to the user's machine. One point is worth noting here: before the query words are processed they are stemmed before they are searched for in the database. The database consists of a number of tables that are arranged so as to facilitate faster retrieval of the data. This database is housed in a database server that is called Search Engine Indices, which is connected to the search engine. The typical English search engines will have more than one database server due to the huge number of English web sites. When the user types the query it is taken to the server of the search engine.

The proposed search engine validates the query and then translates it into the structured query language (SQL) which is understandable to the database and passes this SQL query to the SEDB. The SEDB identifies the database entries that match the query given and sends to the proposed search engine server these entries along with other information related to other entries such as the title, the author name, URL and the matching portion from the content of the corresponding entry. The proposed search engine sorts these database entries using a ranking algorithm. The ranking algorithm determines the relevancy of a retrieved webpage to the user query. The retrieved sites are then displayed along with links to these sites and a small portion of text from the matched content. This text gives an idea to the user about the page before the user goes to that particular page.

Advanced Search options allow the user to search for various combinations of the query terms. Some of the search options include Boolean search and phrasal search. In Boolean Search, several options should be available to the user to refine the query. This is important because the search should return only the relevant pages to the user. Boolean search option includes XOR, NEAR, OR, AND and NOT logic operators, the default being OR

operator. Boolean search can be illustrated by the following example. Consider a query consisting of two words. The search results for the OR logic will retrieve the pages containing either of the two terms and the search results for the AND logic retrieve the pages containing both query terms. The NOT search returns the WebPages that not contain the NOT term.

The search engine automatically searches for both the AND & OR logic. The results of AND search are displayed at the beginning followed by the results of the OR search. Phrasal search looks for a phrase instead of a word in the database. To include phrase search in the query the user should type the phrase between two quotes. The corresponding phrase will be searched as is in the database. This option is particularly useful if the user knows a phrase in the domain of the search. However, this option requires huge processing power and bigger memory in the database. In addition to the use of the advance search mode, the user could specify the retrieved result by the occurrence of the query within the web page, the publishing date of the web page, as it is represented in Figure 5.

7. System Evaluation:

Evaluating of the crawler system is to measure the average of crawled web pages per time unit. In this research the crawling average is 5pages/sec with 5 connections per machine to crawl 26160 web pages. The Effective crawling algorithm is based on best first search technique and the target web page (the most important) is the page that has high PageRank value and the largest number of in-come links. Figure 6 illustrates the diagram of average PageRanke score by hours of crawl.

The average score for pages crawled on the first hour is 8.06; more than four times the average score of 2.03 for pages crawled after ten hours. The average score tapers from there down to 1.17 after twenty hours, 0.82 after 30 hours (more than one day). Clearly, the more high quality (more important) pages are downloaded, i.e., pages with high PageRank, early in the crawl than later on.

Evaluating of the search result is to measure how well the retrieved results meet the user's particular information need. There are two standard measurements Recall and Precision that are used in evaluating the performance of the proposed search engine. Recall and Precision are based on human –relevance judgments and are thus difficult to establish unless such a judgment is readily available. To explain these principles, some examples will be introduced. P is the set of all relevant web pages in existence at a certain point of time. R is the set of all results returned for a search at the same time. C is the set of all relevant results. p, r, and c are defined as the count of the capitalized sets (e.g. p is the amount of elements in P).Recall and Precision could be defined as:

1. Recall

Recall determines the percentage of relevant documents that were retrieved. The Recall value is between 0 and 1. It is defined as:

$$Recall = \frac{Number\cdot of\cdot relevant\cdot documents\cdot retrieved}{Number\cdot of\cdot relevant\cdot documents} = \frac{c}{p}$$

A high recall means the most of the page that should be returned by a perfect search engine is returned. While in normal or in advanced mode all results that are presented in the Search Engine User Interface are still used. Those pages must have a rank score higher than 10% to be

retrieved. The full evaluation of the recall can only be done by doing a user plane review or (by a user judgment).

2. Precision

Precision is a measure that shows how much of what the user sees is relevant. The resulting value is a real number between 0 and 1. Precision is very important to the proposed search engine given thousands of web pages. This measure is defined as:

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Number\ of\ retrieved\ documents} = \frac{c}{r}$$

In this research, five different kinds of queries are tested and evaluated by Recall and Precision of the retrieved results. The results obtained are illustrated in Table.1. The results show that the general Precision of the retrieved web pages is always 100%, which is reasonably good. From the Precision results it is obvious that the rank values of the retrieved web pages reflect the real relevancy of the existent web pages in the proposed search engine databases.

8. Crawling Systems Comparison

Several parameters can be used to compare crawlers of search engines that are listed in Tables 2. Notice that Mercator crawler is used in AltaVista search engine.

9. Conclusions:

1. By implementing the proposed search engine with different kinds of queries, the yielded results prove that the aim of this research is achieved using Best-First Crawl and Distributed Crawling Techniques.
2. By crawling the web pages two problems are detected; *Alternative paths on the same host* (existing of multiple paths to the same file on a given host), and *Replication across different hosts* (multiple copies of a document may reside in different web servers). Both of these problems are solved by avoiding download duplicate documents using Document Fingerprint.
3. The primary copy of a collection may often be updated, while mirror copies are updated only daily, weekly, and monthly. However the mirrors of these collections are usually out of date, depending on how often they are updated.
4. The documents in a collection may *not* themselves appear as exact replicas in another collection. For instance, one collection may have documents in HTML while another collection may have them in Adobe PDF or Microsoft Word. Similarly, the documents in one collection may have additional buttons, links and images that make them slightly different from other versions of the document.

10. Suggestions for future work:

By the experiments, several suggestions are identified that could be implemented in the future to make the research more optimal in its activation with the user:

1. Because of the long time spent in indexing process, a distributed web pages indexing system is suggested.
2. Because of the large storage space that is required by the Inverted Index, a lossless compression method could be used to reduce the storage space.
3. Building an Intelligent spelling checker to identify the wrong word in the user query. This

provides a helpful user interface to the user.

11. References

1. Yong, J. Lee, Ho, S. Lee, Kim, Y.. SCrawler: A Seed – By – Seed Parallel Web Crawler. School of computing, Soongsil University, Seoul, Korea. 2007. Available at: http://dblabssu.ac.kr/publication/LeLe07.pdf
2. Hawking, D., Web Search Engine: Part1 & Part2, CSIRO ICT Center. June 2006. Available at: http://computer.org/portal/site/computer/
3. Andre, L. Barroso, Dean, J., Hölzle, U., Web Search for a Planet: The Google Cluster Architecture. The IEEE Computer Society. 2004. Available at: http://research.google.com/archive/googlecluster-ieee.pdf
4. Abdollahzadeh, A. Barfourosh, H. R. Motahary Nezhad, M. L. Anderson, D. Perlis. Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition.
5. Junghoo, cho. Crawling the web: Discovery and Maintenance of Large-Scale web Data. Ph. D. in Computer Science. November 2001. Available at: http://www.webir.org/resource/phd/cho-2001-thesis.pdf
6. Bhatia, M., Gupta, D., Discussion on Web Crawlers of Search Engine, Proceedings of 2nd National Conference on Challenges & Opportunities in Information Technology (COIT-2008) RIMT-IET, Mandi Gobindgarh. March 29, 2008. Available at: http://wwwrimtengg.com/coit2008/proceedings/WB01.pdf
7. Web Crawling and Indexing, Cambridge University Press. January 25, 2008. Available at: http://www1.cs.columbia.edu/~cs6998/textbook/chapter20-crawling.pdf
8. Andrei, Z. Border, Najork, M., Janet, L. Wiener, Efficient URL Caching for World Wide Web Crawling, May 24, 2003. Available at: http://research.microsoft.com/pubs/65157/p96-broder.pdf
9. Hafri, Y., Gjeraba, C., Dominos: A new Web Crawler's Design, Ecole polytechnique de nates, September 16, 2004. Available at: http://iwaw.europarchive.org/04/Harfi.pdf
10. Shkapenyuk, V., Suel, T., Design And Implementation of a High-Performance Distributed Web Crawler. In ICDE. 2002 . Available at: http://cis.poly.edu/suel/papers/crawl.pdf
11. Junghoo, cho., Garcia-Molina, H., Page, L. , Efficient Crawling Through URL Ordering. In proceeding of the seventh international web conference, Brisbanc, Australia, April 14-18, 1998. Available at: http://www.csd.uchgr/~hy558/papers/cho-order.pdf
12. Selvitri, F., high Performance Issues in Web Search Engines: Algorithms and Techniques. May 2004. Available at: http://www.webir.org/resource/phd/

Silvestri-2004.pdf

13. Molina G. Hector, Searching the Web, August 2001. Available at:

http://oak.cs.ucla.edu/~cho/papers/cho-toit01.pdf

14. Cheng, J.. Design and Implementation of ICS Web Search Engine. Information and Computer Science Department University of California, Irvine. 1996. Available at:

http://contact.ics.uci.edu/download/cheng-report.pdf

15. Bahle, D.. Efficient Phrase Querying. Ph. D. in Computer Science. Royal Melbourne Institute of Technology, Melbourne, Victoria, Australia. March 17, 2003. Available at:

http://portal.acm.org/

Table (1) Searching result, Recall and Precision measures.

Query	Results	Recall	Precision
Web crawling	891 very relevant 23 relevant	90%	100%
"ASP.NET source code"	368 very Relevant 52Relevant	80%	100%
Sport or game	40 Very Relevant 12 relevant	100%	100%
HTML and Java Script	134 Very Relevant 10 Relevant	70%	100%
Internet not web	7 Very Relevant	100%	100%

Table (2) System Comparison Table

Crawlers	Proposed Crawler	Google	UbiCrawler	Mercator	Internet Archieve
urlsEncountered data structure	array of values	—	—	hash-table disk sorted list	Bloom Filter per domain
Programming Language	VB.NET	C++	Java	Java	—
Connection per machine	5	300	4	100	64
System size (# of machines)	2	4	16	4	—
Crawl order	26 thousand	24 million	—	891 million	100 million
Crawl rate (pages/sec)	4	48	52	600	10

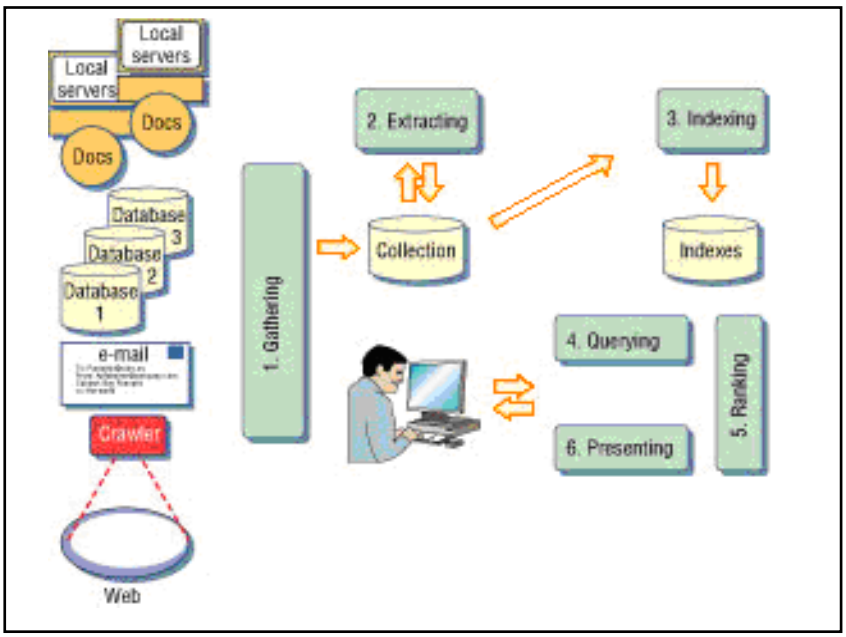

Figure (1) A simple architecture of a generic search engine

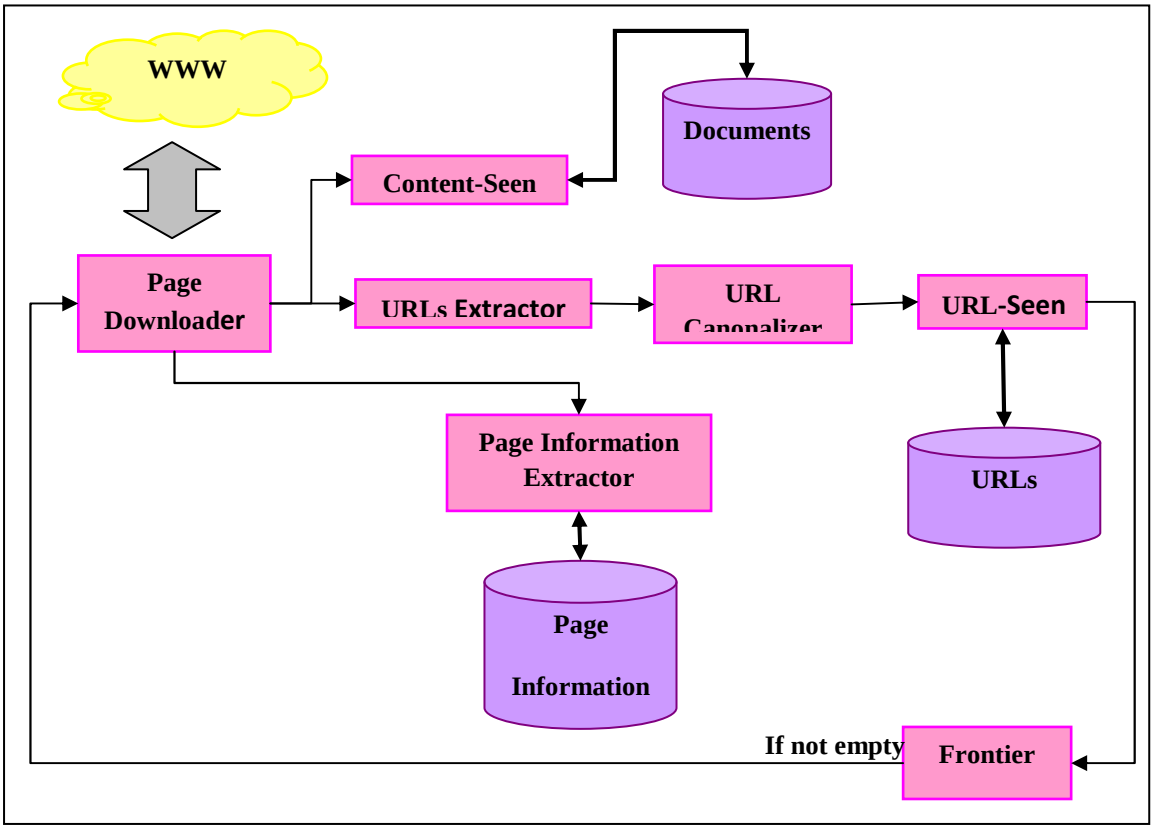

Figure (2) Functional diagram of the Proposed Crawler

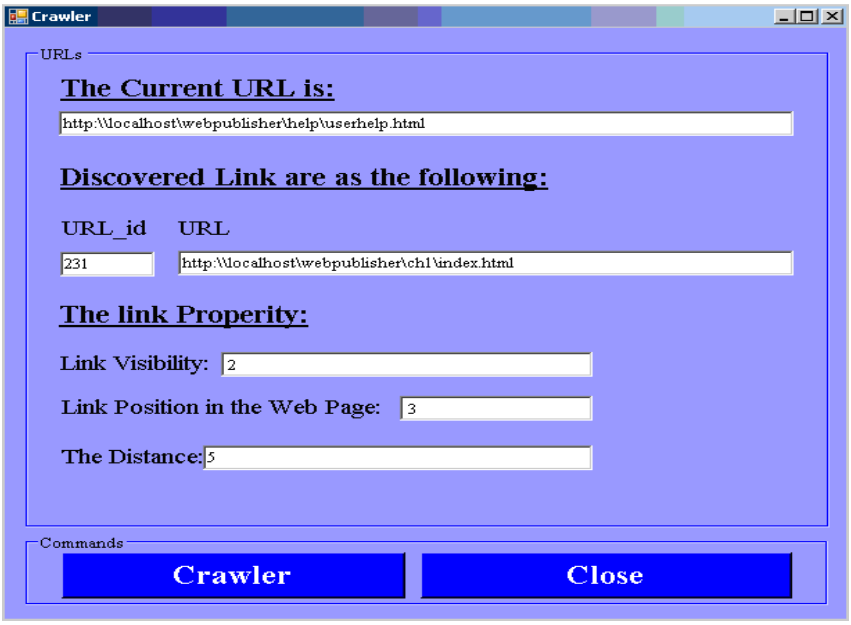

Figure (3) The main interface of the crawling process

Figure (4) Search Engine User Interface in English

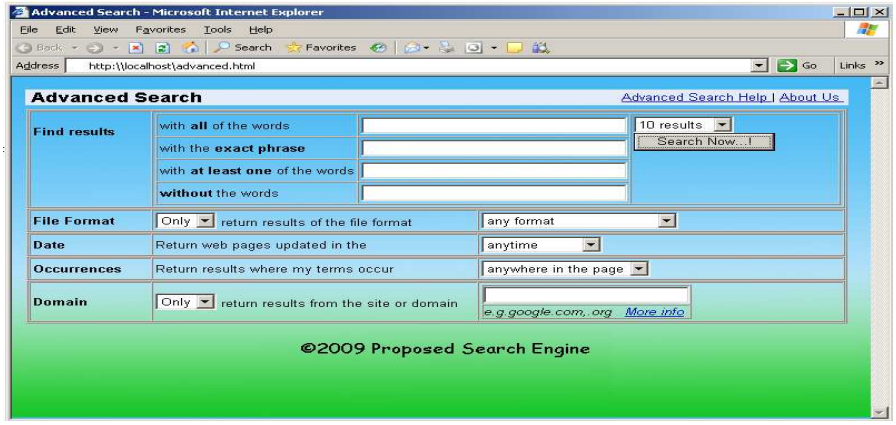

Figure (5) Advanced Mode Search Engine

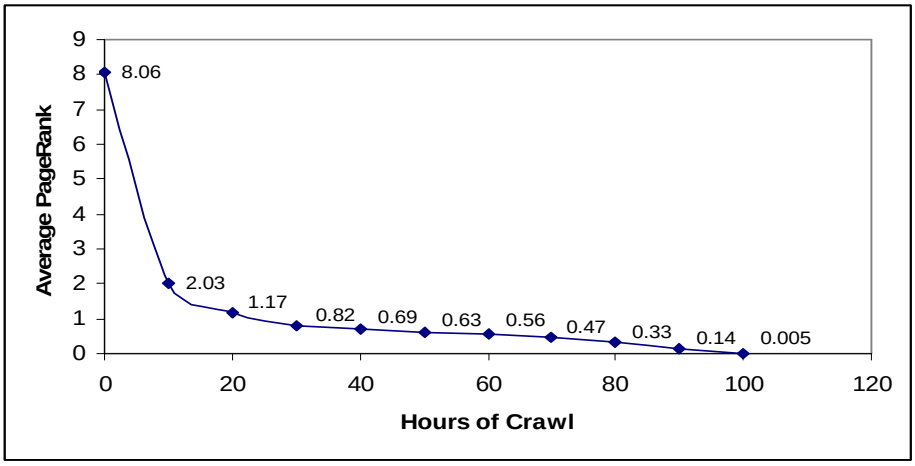

Figure (6) Average PageRank Score by Hours of Crawl