

WEB CRAWLER DESIGN ISSUES: A REVIEW

DEEPIKA*

DR ASHUTOSH DIXIT**

Abstract

The large size and the dynamic nature of the Web increase the need for updating Web based information retrieval systems. Crawlers facilitate the process by following the hyperlinks in Web pages to automatically download a partial snapshot of the Web. While some systems rely on crawlers that exhaustively crawl the Web, others focus on topic specific collections. In present paper the various types of crawlers are discussed. The paper also discusses several web crawler design issues along with their solutions.

Keywords: Web Crawler, types of Crawlers, design issues.

* ASSISTANT PROFESSOR, YMCA UNIVERSITY OF SCIENCE AND TECHNOLOGY, FARIDABAD.

** ASSOCIATE PROFESSOR, YMCA UNIVERSITY OF SCIENCE AND TECHNOLOGY, FARIDABAD.

Introduction

The World Wide Web commonly known as "WWW", "Web" or "W3", is a system of interlinked hypertext documents accessed via the Internet. WWW is growing at very fast rate hence a user has to sift through several pages to come upon the information he/she desires. Search engine help users to find the documents of their interest with the help of its components called crawler. The basic task of a crawler is to fetch pages, parse them to get more URLs, and then fetch these URLs to get even more URLs. It is basically a program that retrieves and stores pages in a repository.

These pages are later analysed by a module called indexer. The crawler typically starts off with an initial set of URLs called, *Seed*. Roughly, it first places *Seed* in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler gets a URL (in some order), downloads the page, extracts any URLs in the downloaded page, and puts the new URLs in the queue. This process is repeated until the crawler decides to stop. The working of a typical crawler is shown in fig 1.

The crawling process is work as follows:

1. The crawler removes the highest-ranked URL from the list of unvisited URLs.
2. The document is retrieved from the Web.
3. A copy of the document is placed in the local repository for indexing by the search engine.
4. The crawler parses the document and extracts the HTML links, with each extracted URL converted to a standardised format.
5. The extracted URLs are compared to a list of all previously extracted URLs ("All URLs" list in Figure 1) and any new URLs are added to this list.
6. If the URL is added to the list of all previously extracted URLs, it is also inserted into the list of unvisited URLs for crawling.
7. The set of URLs is reordered using some scheme [Cho et al., 1998], such as breadth-first ordering or Page Rank [Page et al., 1998], and the process repeats from Step 3 until the set of URLs is empty, the crawl repository is full, or the resources allocated to crawling are exhausted.

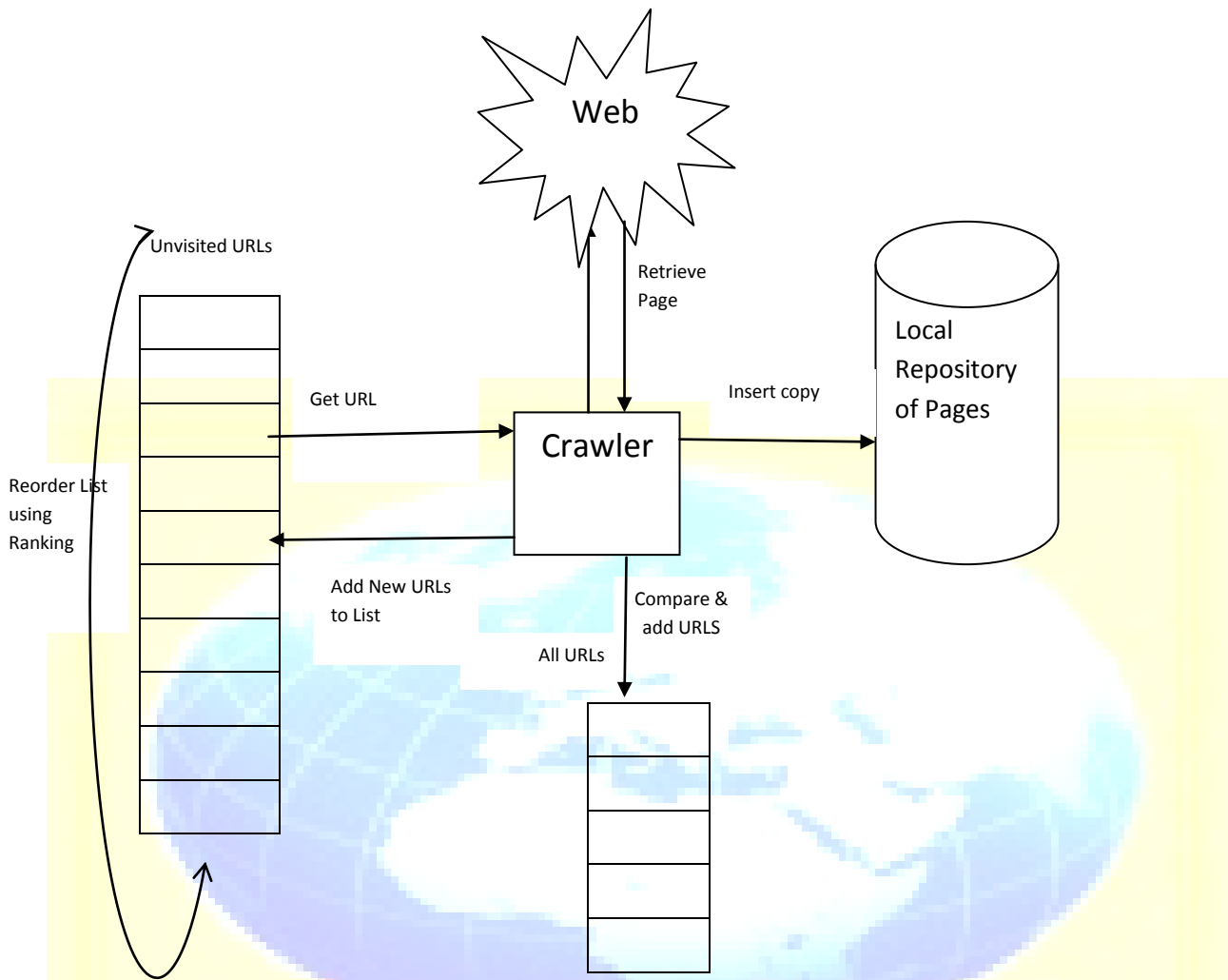


Figure 1 Working of a typical Web Crawler

Related Work

Depending upon their page downloading strategy, the crawlers are of following types:-

1.1 Periodic Crawler

The Periodic Crawler visits the web until the collection has a desirable number of pages, and stops visiting pages. Then when it is necessary to refresh the collection, the crawler builds a new collection by traversing whole web, and then replaces the old collection with the new one. It indexes a new page only after the next crawling cycle starts.

1.2 Parallel Web Crawler

1.2.1 Junghoo Cho: In order to retrieve the whole or significant portion of the web, Parallel Web Crawler runs multiple processes in parallel so that download rate is maximized. A typical parallel crawler tries to fetch more than one page at a time from a given host. Multiple crawling processes of a parallel crawler disperse the network load to multiple regions. In addition to the dispersing load, a parallel crawler may actually reduce the network load.

1.2.2 Parachyd[Sharma et. al.] : In this work modification in the existing working of a parallel crawler has been proposed such that multiple instances of crawler may download/fetch more than one page from a given host. Augmentation of HTML page has been proposed by introducing the table of link and supplying it with every page with different extensions i.e .TOL. This .TOL file provides the list of links embedded in particular HTML page so that in addition to download that page its embedded links also fetch by crawler instances in parallel.

1.3 Focused Crawler

A Focussed crawler [Chakarbarti et. al.] may be described as a crawler, which returns relevant web pages on a given topic. Rather than collecting and indexing all accessible web documents, the goal of a focused crawler is to selectively seek out pages that are relevant to a pre-defined set of topics. A focused crawler analyzes its topic of interest to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web. This leads to significant savings in hardware and network resources.

1.4 Incremental Crawler

In contrast to periodic crawler an incremental crawler [Cho et. al.] updates an existing set of downloaded pages instead of restarting the crawl from scratch each time. It may revisit only the pages that probably have changed, instead of refreshing the entire collection altogether. This optimization may result in substantial savings in network bandwidth and significant improvement in the “freshness” of the collection.

1.5 Distributed Crawler

A distributed crawler [Shkapenyuk et. al.] distributes the downloading task among the downloading instances i.e agents. The number of pages crawled per second per agent are

independent of the number of agents. In other words, it is expected that the throughput to grow linearly with the number of agents. A distributed crawler is a fault tolerant in the sense that it continues to work under *crash faults*, that is, if some agents abruptly die other agents keep working.

Literature Review

Web Crawler Design Issues

The web is growing at a very fast rate and moreover the existing pages are changing rapidly in view of these reasons several design issues need to be considered for an efficient web crawler design. Here, some major design issues and corresponding solution are discussed below:-

➤ How should the crawler get relevant pages to query?

With the increase in web size, the number of applications for processing data also increases. The goal is to take advantage of the valuable information contain in these pages to perform applications such as querying, searching, data extraction, data mining and feature analysis. For some of these applications, notably for searching, the criteria to determine when a page is to be present in a collection are related to the page contents, e.g., words, phrases, etc. However, there are other important situations in which the features of the inner structure of the pages provide better criteria to define a collection than their contents. In view of this issue the following suggestion has been made.

It is suggested by Vidal et. al. that by knowing the structure of a required page beforehand, desired page(s) can be searched more efficiently. So, instead of fetching all pages related to search topic, fetch only those pages which will have similar structure as that of sample page in terms of relevancy.

A tool is developed for generating structure-driven crawlers that requires a minimum effort from users, since it relies on a sample page of the pages to be fetched.

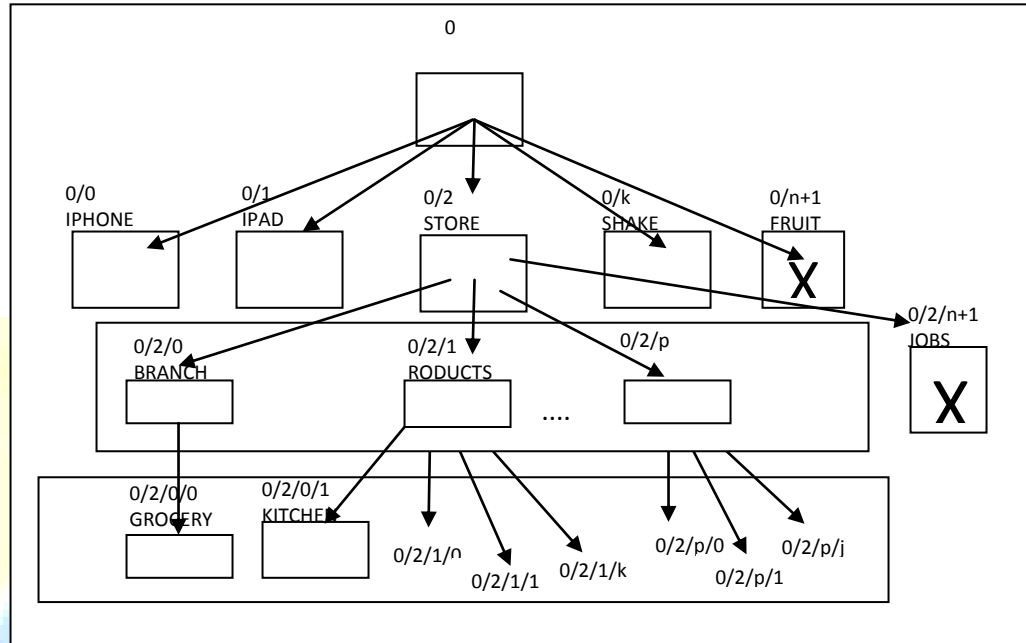


Figure 1.2

To accomplish this, given a sample page and an entry point to a Web site, the tool greedily traverses the Web site looking for target pages, i.e., pages that are structurally similar to the sample page. Next, it records all paths that lead to target pages and generates a navigation pattern which is composed by sequences of patterns of links a crawler has to follow to reach the target pages as shown in figure 1.2. Finally, the tool generates a crawler based on these patterns. From this point on, the crawler can be used to fetch pages that are structurally similar to the sample page, even if new similar pages are added later.

- **How should the crawler refresh pages?** Once the crawler has downloaded a significant number of pages, it has to start *revisiting* the downloaded pages in order to detect changes and refresh the downloaded collection. Because Web pages are changing at very different rates, the crawler needs to carefully decide what page to revisit and what page to skip, because this decision may significantly impact the “freshness” of the downloaded collection. For example, if a certain page rarely

changes, the crawler may want to revisit the page less often, in order to visit more frequently changing ones. There are several approaches to predicting when a document is likely to change and therefore require recrawling. Following are the suggestions:-

One method [Dixit et. al.] is to examine how frequently a document has changed in the past, that is, by examining the past change history. While there are many studies that have examined recrawl frequency when there is a long period of past change history available, this information is not always available, particularly when a document has only been crawled a few times. Little work has investigated schemes that work well when past change information is unavailable or limited.

Another approach for predicting when documents are likely to change is to use the Last-Modified and Expires HTTP headers that are returned along with web documents. When they are properly maintained and accurate, these headers can be a very effective method of improving the freshness of crawled documents. These headers, however, are not always available, and may be deliberately inaccurate to force web clients to update their locally cached copy of documents.

Another efficient approach for computing revisit frequency is being proposed. Web pages which frequently undergo updation are detected and accordingly revisit frequency for the pages is dynamically computed.

Another novel approach is present for maintaining freshness, which uses the anchor text linking documents to determine the likelihood of a document changing, based on statistics gathered during the current crawl. He shows that this scheme is highly effective when combined with existing stateless crawl ordering schemes. This approach allows the crawling scheme to improve both freshness and quality of a collection by combining it with a scheme such as PageRank.

➤ **How should the resources that are enormous be located and tracked?**

There are a large number of manually maintained virtual libraries available on the WWW. Some of these are subject-oriented while others are service oriented. Many VL are specific to a single subject, for example a list of Computer Science Departments. Most of the well-known WWW VL are maintained and updated by one person. Because information is not inserted directly by information providers these systems do not scale well to an Internet with millions of computers. Furthermore, most of these systems are not hierarchical, which also severely impacts scalability. In view of this issue following tools are generated:-

GENVL [McBryan et. al.] is an interactive user-driven hierarchical virtual library system for cataloguing Web resources. The real power of GENVL comes from the built-in recursion which is the key to extendibility and to avoiding the generation of massive linear lists.

| | |
|--------|--|
| Title | Used to provide a hypertext reference to the entry |
| Image | A user supplied in-lined image used in a report |
| Types | Specifies whether a Pointer or a Report |
| Report | Contains the report in the case Type=report |
| Author | Name of the person adding the entry |

Table1.3 Characteristics with a VL Entry

WWW - the WWW Worm - is a resource location tool. It is intended to locate almost all of the WWW-addressable resources on the Internet, and provide a powerful search interface to those resources. Searches can be performed on document titles, reference hypertext, or within the components of the URL name strings of documents.

| | | |
|---|------|----------------|
| T | HTML | Title |
| R | URL | HTML Reference |

| I | URL | HTML | Reference |
|---|-----|------|-----------|
|---|-----|------|-----------|

Table1.4 WWW Archive Format

The T lines denote an HTML file encountered and opened for reading during a search and the second argument is then the URL of the HTML. The remainder of the line is the Title as taken from the *<title>* line required at the top of every HTML. Some files omit a title and in these cases the Title "Zero Length Title" is substituted.

The R line is used to denote a URL referenced in an HTML file which was opened for reading. In this case the second argument is the URL, and the third argument is the URL of the HTML file that referenced the URL. The rest of the line is then the hypertext anchored by the URL reference. The I line is used to represent an in-lined image. In-lined images have no associated hypertext. In these cases we associate only the title string of the containing HTML.

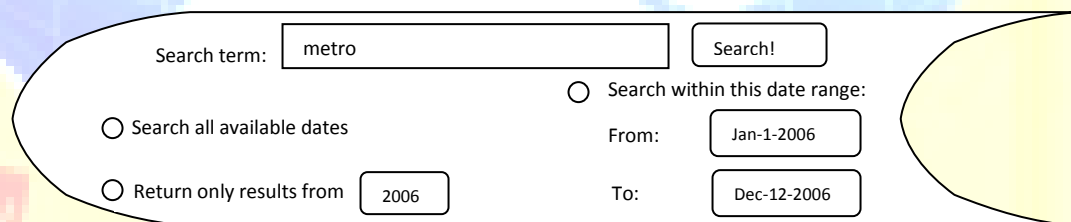
- **How should the crawling process be parallelized?** Due to the enormous size of the Web, crawlers often run on multiple machines and download pages in parallel. This parallelization is often necessary in order to download a large number of pages in a reasonable amount of time. This generates a continuous stream of new URLs of documents to be downloaded and it is clear that the associated work-load can only be served efficiently with proper parallel computing techniques. The incoming new URLs have to be organized by a priority measure in order to download the most relevant documents first.

An efficient and scalable strategy is proposed which consider intra-node multi-core multithreading on an inter-nodes distributed memory environment, including efficient use of secondary memory. The incoming new URLs have to be organized by a priority measure in order to download the most relevant documents first. Efficiently managing them along with other synchronization issues such as URLs downloaded by different processing nodes.

➤ **How should crawler get time sensitive information?**

Time Sensitive Searching is an issue that need to be addressed to get the time sensitive information from the web. Usually Search engines crawl the web and take vast snapshots of site content. As previous crawls are not archived so search results pertain only to a single, recent instant in time. As a result when users request some pages which require past data then search engines are unable to provide because it is not possible to search files that represents snapshot of the web over time.

A temporal search engine [Efendioglu et. al] has been proposed that indexes Internet Archive crawl data in order to provide search results spanning user specified time ranges. It can generate graphs showing query result hit counts across a given time span and even side-by-side comparisons of different query results. These graphs can be used to, among other things, track a term's popularity over time for marketing or academic research purposes. Via a web interface, a user can input any textual query, using either the simple or advanced search functionality as shown in Figure 1.5



The image shows a search interface with the following elements:

- Search term: metro
- Search! button
- Radio button selected: Search within this date range:
- From: Jan-1-2006
- To: Dec-12-2006
- Radio button: Search all available dates
- Radio button: Return only results from 2006

Figure1.5 Chronica's general search interface

Conclusion

Crawler is an important component of a search engine from page downloading point of view. Due to explosive size of the web, a crawler needs to be designed very carefully. In this paper some prominent types of web crawler and their design issues and corresponding suggestion have been discussed.

References

- A.K. Sharma, J.P. Gupta, D. P. Aggarwal, “PARCAHYD: An Architecture of a Parallel Crawler based on Augmented Hypertext Documents”
- Ashutosh Dixit ,A. K. Sharma, “A Mathematical Model for Crawler Revisit Frequency” in the proceedings of IEEE 2nd International Advance Computing Conference (IACC 2010) February 19-20, 2010. Thapar University Patiala
- J. P. Lage, A. S. da Silva, P. B. Golgher, and A. H. F. Laender, “Automatic generation of agents for collecting hidden web pages for data extraction” , Data and Knowledge Engineering, 49(2):177–196, 2004.
- Junghoo Cho and Hector Garcia-Molina, “Incremental crawler and evolution of web”, Technical Report, Department of Computer Science, Stanford University.
- Junghoo Cho, “Parallel Crawlers”, In proceedings of WWW2002, Honolulu, hawaii, USA, May 7-11, 2002. ACM 1-58113-449-5/02/005.
- M´arcio L.A. Vidal, Altigran S. da Silva, Edleno S. de Moura, Jo˜ao M. B. Cavalcanti, “GOGETIT!: a Tool For Generating Structure-Driven Web Crawler”
- Mauricio Marin Rodrigo Paredes Yahoo! Research Latin America Santiago,Chile Carolina Bonacic ArTeCS, Complutense University Madrid, Spain. “High-Performance Priority Queues for Parallel Crawlers”
- Vladislav Shkapenyuk and Torsten Suel, “Design and implementation of a high performance distributed web crawler” , In IEEE International Conference on Data Engineering (ICDE), 2002.
- Oliver A. McBryan, “GENVL and WWW: Tools for Taming the Web.”
- S. Chakrabarti, M. van der Berg, and B. Dom, “Focused crawling: a new approach to topic-specific web resource discovery,” in Proc. of the 8th International World-Wide Web Conference (WWW8), 1999
- Deniz Efendioglu, Chris Faschetti, Terence Parr “Chronica: A Temporal Search Engine” ICWE '06