ASSEMBLING A WEB-CRAWLER

Jitali Patel[1], Varun Raval[2]

[1]Assistant Professor, [2]Student, Dept of CSE, Institute of Technology,
Nirma University, Ahmedabad, (India)

ABSTRACT

Web crawlers are nowadays most important part of the internet. Can we imagine our life without GOOGLE, YAHOO and Bing? The web-crawlers are heart of all those search engines. Besides just web-crawler, many complex technologies such as page ranking system, effective key search techniques, disabling malicious sites and much more are involved in making a search engine. This essay is dedicated for complete understanding of a basic web-crawler and some of its requirements. Keep in mind that for commercial web crawlers, both hardware and software requirements are much more than what is described over here.

Keywords: Web Pages, Web Crawler

I. INTRODUCTİON

A web crawler is a set of algorithms that scans all the websites available on the internet mainly for the purpose of retrieving and updating information related to a specific topic/s. Web-Crawler is mainly used by search engines for updating and retrieving new information to give accurate results of user search. The first inspiration to make web-crawler in mid 1990s was due to the fact that the number of IP addresses grew more than 1 million and it was a big trouble to get latest information on required topic which was made possible by use of web-crawler, key search and text retrieval techniques. How does a crawler fetch "all" Web pages? Before the advent of the Web, traditional text collections such as bibliographic databases and journal abstracts were provided to the indexing system directly, say, on magnetic tape or disk. In contrast, there is no catalog of all accessible URLs on the Web. The only way to collect URLs is to scan collected pages for hyperlinks to other pages that have not been collected yet. This is the basic principle of crawlers. They start from a given set of URLs, progressively fetch and scan them for new URLs (outlinks), and then fetch these pages in turn, in an endless cycle[1]. New URLs found thus represent potentially pending work for the crawler. The set of pending work expands quickly as the crawl proceeds, and implementers prefer to write this data to disk to relieve main memory as well as guard against data loss in the event of a crawler crash. There is no guarantee that all accessible Web pages will be located in this fashion; indeed[2], the crawler may never halt, as pages will be added continually even as it is running. Apart from outlinks, pages contain text; this is submitted to a text indexing system. In this essay, I will be guiding you on the steps required to make a basic web-crawler.

II. CRAWLING THE WEB

Initially we start crawling with a set of URLs. Even a single URL can be used to start a crawl. The main steps of any crawler are:

1. Fetching IP address
2. Parsing page to get required information

After fetching IP address, page is downloaded and analyzed. From this URL page, we get URLs that this page is pointing at and this new list of URL will again be converted to list of IP addresses, downloaded and analyzed. This is the general procedure of any web crawler whose individual steps are explained in detail below.

DNS Catch and Prefetch

Any web page URL is composed of two types of information. Host name and page destination within that host. E.g. en.wikipedia.org/wiki/main_page, here en.wikipedia.org is host name and page destination is /wiki/main_page. DNS (Domain Name Server) is used to obtain IP using host name or server name i.e. en.wikipedia.org in our example. Steps to maintain database are described below:

1. All new pages fetched from visited pages are added to a list of unvisited pages
2. As need arises (not enough pages to download), these new pages in sequence will be added to a DNS queue that is maintained locally.
3. This DNS queue will be used by crawler to fetch IP address from DNS server.
4. After fetching IP address, these URLs including IP address and page destination e.g. 208.80.154.224/wiki/cricket will be added to a list from where downloader will download pages.

A desktop PC with 256 MB of RAM and a disk cache of a few GB will be adequate for a caching DNS, but it may help to have a few (say, two to three) of these.

Multiple Concurrent Fetches

To save time and to maximize use of any good processing computer, concurrent fetches can be made using extensive use of multithreading and binding more than one socket to DNS server for retrieval of multiple IP addresses at single time. Speed of IP fetches will be dependent upon the number of sockets allocated, processing speed of computer and speed of internet connection[3].

Parsing page

Parsing page means retrieving needed information from a page like metadata, hyperlinks, text attached to hyperlinks, title of page, etc. Metadata include information like expiry date of that page, last update information, type of contents in that page and similar things. Metadata is used to decide when we should revisit that same page for updating our database. For parsing a page, parsers such as DOM, SAX and STAX can be used. These parsers can download the whole page in html format so that any page can be revisited locally. Parsers are mainly used by web-crawlers to search for hyperlinks in page and add them to fetch list and get as much information from metadata about the page as possible[5].

Link Extraction and Normalization

While parsing a page we encounter two types pf URLs:

1. Absolute URL
2. Relative URL

Absolute URLs are the URL which are in their complete form. E.g. en.wikipedia.org/wiki/cricket_ball. This URL can directly be included in the list of URL for fetching IP addresses. But relative URLs are having just page reference relative to the current host. E.g. /wiki/cricket_ball. These types of URLs are needed to be converted to normal form before adding to fetch list.

Robot Exclusion

Many websites have sensitive information on their pages such as account information and other confidential information and these websites don't want the crawlers to crawl such pages. So, Next step is to check whether the server restricts crawling a URL using the robots.txt mechanism. This file is usually found in the Http root directory of the server (such as en.wikipedia.org/robots.txt). This file specifies a list of pages that crawlers should *not* attempt to fetch. The robots.txt file is meant for crawlers only and does not apply to ordinary browsers. If anyone tries to crawl such pages than our IP address may be restricted by that server.

Eliminate already existed URLs

Each and every time a new URL is encountered while parsing a page, it is checked against the list of URLs whose IP addresses are already fetched. For this reason we are needed to maintain a table which will include a list of URLs whose IP addresses have been fetched. Each and every time IP of any address is fetched it will be added to this table. In this way, we can eliminate already existed URLs. But sometimes crawling already visited pages may prove useful. There two types of web-pages

1. Static
2. Dynamic

This information about any page can be obtained from the "last updated" information in meta tags of that page. If last updated information is later than our last fetch of that page, we must refetch that page to update our database.

After Parsing a Page

After parsing a page it should be removed from the list containing a pages to download and added to a list of already downloaded pages just for sake of maintaining a record of downloaded pages. After parsing a page if number of pages to be downloaded are less than certain threshold value (not enough pages to download) than if process for fetching of IP is waiting for a signal to start fetch, it should be notified[4].

Robots.txt

Many websites have sensitive information on their pages such as account information and other confidential information and these websites don't want the crawlers to crawl such pages. For this reason, all the servers contain a file in their root directory called robots.txt. robots.txt file contain information about pages which are disallowed by the server to crawl. If anyone tries to crawl such pages than our IP address may be restricted by that server.

Fetch at some max rate

It is advisable to be kind to any server in terms of rate of fetching information. Any server imposes a limit on the number of requests from any IP address in a given time interval. This number is dependent on individual servers. Thus, before downloading any page we must check if that page's URL is not in the list of busy servers. If that page is in the list, we must wait until the sockets using that page end their connection and go further leaving that page to download in future. If that page is not in the list, we can stat downloading and should see weather that server has now become a busy server and if so, it must be added to the list of busy servers.

III. TEXT REPOSITORY

Many web-crawler like to maintain a text repository of crawled web pages especially for the webpages that are static. These local repository will be used for performing offline operations such as page ranking, giving

prompts to user based on user entered keyword, etc. Making a text repository will be beneficial if space constraint is not there, internet bandwidth is more and pages to be downloaded are static.

Conclusion

Following the above mentioned steps in programming language of your choice, you will be able to build a small scale web crawler which is able to fetch DNS addresses at some hundred pages per sec. rate and download as per your internet speed and type of text you want to download. A good multithreaded program involving use of multiple sockets in today's 5^{th} gen computers can help crawl pages at faster rate. In order to go for steps to build a search engine, you will need to add several functionalities like page ranking system, key search algorithms, congestion control techniques and obviously a large data space is required.

REFERENCES

[1]. Chakrabati, Soumen.Mining The Web. Bombay: Morgan Kaufmann Publishers

[2]. Markov, ZDRAVKO. Data Mining The Web. New Britain: Wiley Interscience Publications

[3]. Liu, Bing. Web Data Mining. Chicago, USA: Springer-Verlag Berlin Heidelberg

[4]. Rungsawang, Arnon, and Niran Angkawattanawit. "Learnable topic-specific web crawler." Journal of Network and Computer Applications 28, no. 2 (2005): 97-114.Zhang, Huaxiang. An online semi-supervised clustering approach to topical web crawlers. Science Direct (2010): Pages 491 - 495.

[5]. Stevanovic, Dusan, Aijun An, and Natalija Vlajic. "Feature evaluation for web crawler detection with data mining techniques." Expert Systems with Applications 39, no. 10 (2012): 8707-8717.