Implementation of Web Crawler on Client Side

Thae Nu Aung
Computer University (Mandalay)
thaenuaung.ucsm@gmail.com

Abstract

Web Crawler forms the back-bone of applications that facilitate Web information retrieval. Generic crawlers and search engines are like public libraries: they try to cater to everyone, and do not specialize in specific areas. In this paper, we have presented the architecture and implementation details of our crawling system which can be deployed on the client machine to browse the web concurrently and autonomously. We have implemented the Crawler that can crawl and index pages from any Website. This demonstrates the working of a web crawler developed in java. Finally, Our Crawler will allow us to launch crawling process at maximum speeds without concern for the state of servers and usual network latency time.

1. Introduction

There is no doubt that the Internet is making a wide influence on every aspect of modern society including person's life style and its structure. With the explosive growth of the World Wide Web in the last 15 years, the task of processing and analyzing Web-scale resources has become much more complex. The World Wide Web has grown at a phenomenal pace, from several thousand pages in 1993 to over 3 billion today. This explosion in size has made Web Crawlers indispensable for information retrieval on the Web. So, today's search engines are equipped with specialized agents known as Web Crawlers dedicated to crawling large Web contents online. These contents are then analyzed, indexed and made available to users. Crawlers interact with thousands of Web Servers over periods extending from a few weeks to several years. A web Crawler is a program or automated scripts which browse the World Wide Web in a methodical automated manner [1]. The end result of crawler is a collection of web pages, HTML or plain text at a central location. The collection policy implemented in the crawler determines what pages are collected, and which of those pages are indexed [2]. The First Crawler, Matthew Gray’s wanderer, was written in spring of 1993, roughly coinciding with the first release of NCSA Mosaic [3].

2. Theory Background

2.1. Web crawler

These are robots that traverse the Web, navigating through hyperlinks and performing a job, such as recognizing and indexing resources. Such Web Crawling robots are known by a variety of aliases, including crawlers, trawlers, spiders and worms. Today, web crawlers are an essential component of all search engines and are increasingly becoming important in data mining and other indexing applications. Web Crawlers are mainly used to index the links of all the visited pages for later processing by a search engine. Such search engines rely on massive collections of web pages that are acquired with the help of web crawlers, which traverse the web by following hyperlinks.

2.2. How a crawler works

Running a web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers, which are all beyond the control of the system. Web crawling speed is governed not only by the speed of one’s own Internet connection, but also by the speed of the sites that are to be crawled. Especially if one is a crawling site from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel. Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. Following is the process by which Web crawler’s work [4]:

1. Download the Web page.

2. Parse through the downloaded page and retrieve all the links.
3. For each link retrieved, repeat the process.

The Web crawler can be used for crawling through a whole site on the Inter-/Intranet. You specify a start-URL and the Crawler follows all links found in that HTML page. This usually leads to more links, which will be followed again, and so on. A site can be seen as a tree-structure, the root is the start-URL; all links in that root-HTML-page are direct sons of the root. Subsequent links are then sons of the previous sons.

A single URL Server serves lists of URLs to a number of crawlers. Web crawler starts by parsing a specified web page, noting any hypertext links on that page that point to other web pages. They then parse those pages for new links, and so on, recursively. WebCrawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve web pages at a fast enough pace. A crawler resides on a single machine. The crawler simply sends HTTP requests for documents to other machines on the Internet, just as a web browser does when the user clicks on links. All the crawler really does is to automate the process of following links.

Web crawling can be regarded as processing items in a queue. When the crawler visits a web page, it extracts links to other web pages. So the crawler puts these URLs at the end of a queue, and continues crawling to a URL that it removes from the front of the queue [5].

2.3. Design details

A crawler for a large search engine has to address two issues [6]. First, it has to have a good crawling strategy i.e., a strategy to decide which pages to download next. Second, it needs to have a highly optimized system architecture that can download a large number of pages per second while being robust against crashes, manageable, and considerate of resources and web servers. In this paper, the model of a crawler is implemented. It is used on the client side with a simple PC, which provides data to any search engines as other crawlers provide.

A high level architecture of a web crawler [7] has been analyzed as in figure 1 for building web crawler system on the client machine. Here, the multi-threaded downloader downloads the web pages are decomposed into URLs, contents, title etc. The URLs are queued and sent to the downloader using some scheduling algorithm. The downloaded data are stored in a database.

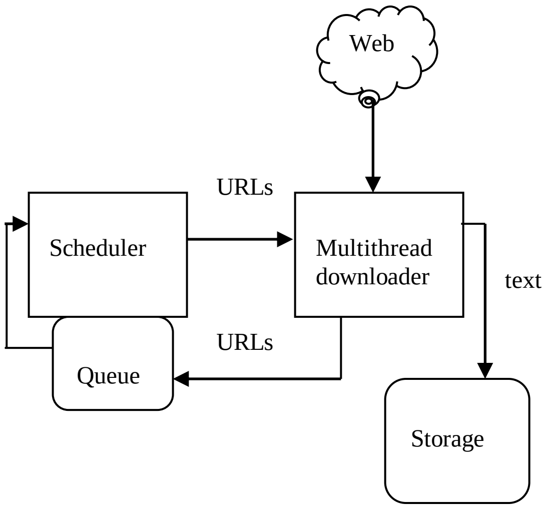

Figure 1. High level architecture of a standard web crawler

2.4. Breadth-first indexing

The idea of breadth-first indexing is to retrieve all the pages around the starting point before following links further away from the start. This is the most common way that robots follow links. If your robot is indexing several hosts, this approach distributes the load quickly, so that no one server must respond constantly. It's also easier for robot writers to implement parallel processing for this system. e.g. limiting crawling processes to within a site, or downloading the pages deemed most interesting first.

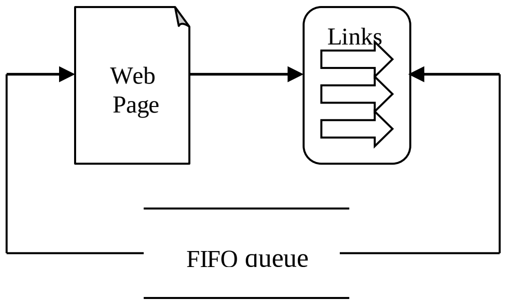

Figure 2. Breadth-first crawler

2.5. Web crawler Algorithm

Begin

```
Step 1: Breadth-First (starting_urls)
          //accepts the URL  address as input//
Step 2: for each url (starting_urls) do
           Enqueue(frontier,url);
          //stores the list of URL links to download in
           frontier //
Step 3: while (visited < max_pages && frontier not
          Empty) do
          //run the algorithm until the maximum
pages
          are reached or frontier is empty//
          url := dequeue(frontier);
          //URL in the frontier is fetched//
        page := fetch(url);
        visited := visited +1;
        //visited pages are counted to prevent the
        infinite loops//
        enqueue(frontier,extract_links(page));
        //URL links in the page are extracted and
        restored in the frontier with queue order//
End
```

3. System Architecture

We have implemented the web crawler application into four main components :

(1)Controller

(2)Fetcher

(3)Parser and

(4)Queue.

First component, we have designed Controller to focus on the graphical user interface (GUI) designed for the web crawler and for controlling the operations of the crawler. The GUI enables the user to enter the start URL, enter the maximum of URLs to crawl, view the URLs that are being fetched. It controls the fetcher and parser. Second component, Fetcher has been designed to start by fetching the page according to the start URL specified by the user. It also retrieves all the links in a particular page and continues doing that until the maximum number of URLs is reached. It is a high performance asynchronous HTTP client capable of downloading hundreds of web pages in parallel. We use multi-thread and asynchronous fetcher. Third component, Parser has been designed to parse the URLs fetched by the fetcher and saves the contents of those pages to the database. Other functions such as discarding unnecessary items and restoring relative hyperlink to absolute hyperlink are also to be taken care of by the parser. Final component, Queue stores the URLs produced by the Fetcher and sends to the Controller. The architecture of our system is shown in figure 3.

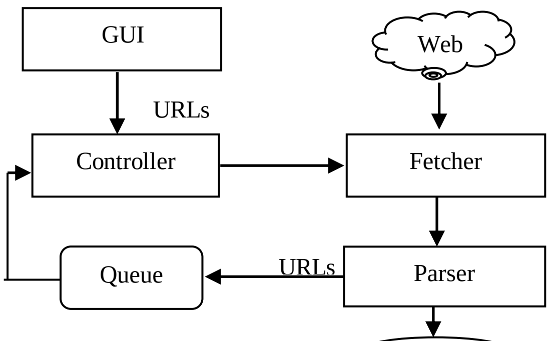

Figure 3. System architecture

3.1. Keyword searching

This searching is taken from the user as input and crawl html links by searching the keyword from the database and gives the indexing results to the user. A simple browser is designed to allow user to browse the pages directly from the application, instead of using a browser outside of the system.

4. Implementation

4.1. Development Environment

To implement the crawler, Java programming language was chosen because of mainly three reasons:

1. It is one of the most well-known programming languages.

2. There are a lot of sample codes and modules worth referring.

3. This programming is easy to maintain and develop separately.

4. Java's combination of features that including threads, garbage collection, objects, and exceptions, made our implementation easier and more elegant.

Furthermore, at final stage of this development, the crawler can be implemented as a form of Java Applet so that the program can run on various platforms without any limitation. Therefore, Java programming ability was required for this implementation.

4.2. Implementation of the system

In figure 4, system flow diagram of our crawler is presented. The currently proposed web crawler uses breadth first search crawling to search the links. Breadth first crawling checks each link on a page before proceeding to the next page. Thus, it crawls each link on the first page and then crawls each link on the first page's first link, and so on, until each level of links has been exhausted. The proposed web crawler is deployed on a client machine. User enters the URL for example http://mandalaydirectory.com in the browser created. Once the breadth-first search crawl button is pressed, an automated browsing process is initiated. The HTML page contents of mandalaydirectory.com homepage are given to the parser. The parser puts it in a suitable format and the list of URLs in the HTML page are listed and stored in the queue. The URLs are picked up from the queue and each URL is assigned to a fetcher. After the page is downloaded, it is added to the database and then the fetcher is set as free. The application remembers which pages it's already visited, so it won't search any web page twice. This prevents infinite loops. As the user inspects the list of URLs, he can see that the application performs a breadth-first search. In other words, it accumulates a list of all the links that are on the current page before it follows any of the links to a new page. If the user let the tour run without stopping, it will eventually stop on its own once it's found 20 files. The user has a choice to stop the search process at any time if the desired results are found. The implementation details are given table 1.

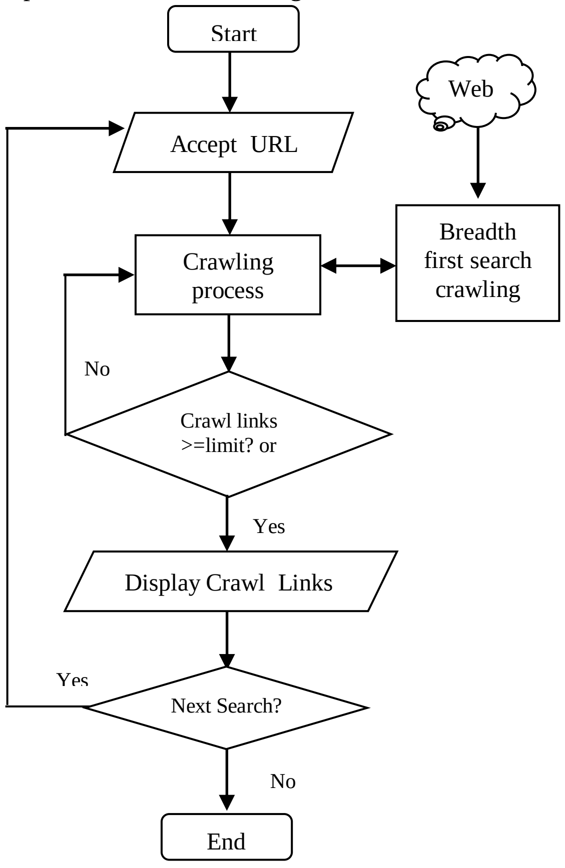

Figure 4. System flow diagram of crawling stage

Feature	**Support**
Help manual	No
Integration with other application	No
Specifying start URL	Yes
Support for Breadth First Crawling	Yes
Support for Depth First Crawling	No
Support for Broken Link Crawling	No
Support for Archive Crawling	No
Check for validity of URL specified	Yes

Table 1. Functionality of the web crawler application

5. Conclusion

This system is the implementation of a basic search crawler application for searching the web and it illustrates the fundamental structure of crawler-based applications. With this application you can search the web, in real time, URL by URL, looking for matches to the criteria. This application is developed using java because of the various features that support WebCrawler application. Mandalay Directory Website is implemented as an example to crawl.

5.1. Benefits of the system

Our system saves user's time when they search instead of trying to guess at a path of links from page to page. Time saving is especially important given the increase in the size and scope of the Web. Web crawler makes the web more friendly and useful tool. It is useful because it can provide some context for a searcher's particular query: by issuing a well-formed query, a searcher can find the breadth of information about the particular topic and can use that information to further refine his goal. Searchers frequently issue a broad query which they refine as they learn more about their intended subject.

5.2. Limitation and further extensions

This system cannot be done the integration with other applications. A major open issue for future work is a detailed study of how the system could become even more distributed, retaining through quality of the content of the crawled pages. Due of dynamic nature of the Web, the average freshness or quality of the page downloaded need to be checked, the crawler can be enhanced to check this and also detect links written in VB scripts and also provision to support file formats like XML, RTF, PDF, Microsoft word and Microsoft PPT can be done. Future work in crawling can be done by implementing different other crawling techniques.

6. References

[1] Baden Hughes , "Web Crawling" , Department of Computer Science and Software Engineering, University of Melbourne.(www.csse.unumelb.edu.au)

[2] Brian Pinkerton , “Web crawler: Finding what people want”, University of Washington, 2000.

[3] Carlos Castillo , “Effective Web Crawling”, Department of Computer Science, University of Chile, November 2004.

[4] Garcia-Molina, Hector, "Searching the Web", August 2001.

[5] Monica Peshave and Advisor:Kamyar Dezhgosha, "How search engines work and a web crawler application", Department of Computer Science, University of Illinois at Springfield.

[6] Internet Growth and Statistics: Credits and Background.
http://www.mit.edu/people/mkgray/net/background.html

[7] The Web Robots Pages.
http://info.webcrawler.com/mak/projects/robot/robots.html