Web Crawler On Client Machine

Rajashree Shettar, Dr. Shobha G

Abstract- The World Wide Web is a rapidly growing and changing information source. Due to the dynamic nature of the Web, it becomes harder to find relevant and recent information.. We present a new model and architecture of the Web Crawler using multiple HTTP connections to WWW. The multiple HTTP connection is implemented using multiple threads and asynchronous downloader module so that the overall downloading process is optimized.

The user specifies the start URL from the GUI provided. It starts with a URL to visit. As the crawler visits the URL, it identifies all the hyperlinks in the web page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited and it stops when it reaches more than five level from every home pages of the websites visited and it is concluded that it is not necessary to go deeper than five levels from the home page to capture most of the pages actually visited by the people while trying to retrieve information from the internet.
The web crawler system is designed to be deployed on a client computer, rather than on mainframe servers which require a complex management of resources, still providing the same information data to a search engine as other crawlers do.

Keywords: HTML parser, URL, multiple HTTP connections, multi-threading, asynchronous downloader.

I. Introduction

1.1 Working of a general Web crawler

A web crawler is a program or an automated script which browses the World Wide Web in a methodical automated manner. A Web crawler also known as a web spiders, web robots, worms, walkers and wanderers are almost as old as the web itself [1]. The first crawler, Matthew Gray's wanderer, was written in spring of 1993, roughly coinciding with the first release of NCSA Mosaic [5]. Due to the explosion of the web, web crawlers are an essential component of all search engines and are increasingly becoming important in data mining and other indexing applications. Many legitimate sites, in particular search engines, use crawling as a means of providing up-to-date data. 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 and storing downloaded pages in a large database that is later indexed for efficient execution of user queries. Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. Following is the process by which Web crawlers work [6]:

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. Web-crawler 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. (Garcia-Molina 2001).

1.2 Resource Constraints

Crawlers consume resources: network bandwidth to download pages, memory to maintain private data structures in support of their algorithms, CPU to evaluate and select URLs, and disk storage to store the text and links of fetched pages as well as other persistent data.

II. Design Details

A crawler for a large search engine has to address two issues [2]. First, it has to have a good crawling strategy i.e. a

Manuscript received January 25, 2008.
Rajashree Shettar is working as Asst. Professor in the Department of Computer-Science at R.V. College of Engineering, Mysore Road, Bangalore-560059, Karnataka, India. Phone: +91-080-28601874; fax: +91-080-28601874; e-mail: (rajshri99@ yahoo.co.in).
Dr. Shobha. G is working as Professor in the Department of Computer-Science at R.V. College of Engineering, Mysore Road, Bangalore-560059, Karnataka, India. Phone: +91-080-28601874; fax: +91-080-28601874; e-mail: (shobhatilak@ rediffmail.com).

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 we present a model of a crawler on the client side with a simple PC, which provides data to any search engines as other crawler provide. To retrieve all webpage contents, the HREF links from every page will result in retrieval of the entire web's content

- Start from a set of URLs
- Scan these URLs for links
- Retrieve found links
- Index content of pages
- Iterate

The crawler designed has the capability of recursively visiting the pages. The web pages retrieved is checked for duplication i.e. a check is made to see if the web page is already indexed if so the duplicate copy is eliminated. This is done by creating a data digest of a page (a short, unique signature), then compared to the original signature for each successive visit as given in figure 3. From the root URL not more than five links are visited and multiple seed URLs are allowed. The indexer has been designed to support HTML and plain text formats only. It takes not more than three seconds to index a page. Unusable filename characters such as "?" and "&" are mapped to readable ASCII strings. The WWW being huge, the crawler retrieves only a small percentage of the web.
We have considered two major components of a crawler - collecting agent, and searching agent [3]. The collecting agent downloads web pages from the WWW and indexes the HTML documents and storing the information to a database, which can be used for later search. Collecting agent includes a simple HTML parser, which can read any HTML file and fetch useful information, such as title, pure text contents without HTML tag, and sub-link.

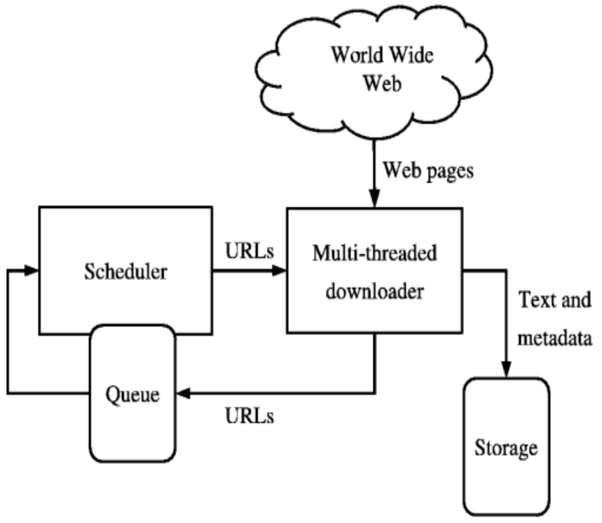

Figure 1: High-level architecture of a standard Web crawler.

The searching agent - **s**earching agent is responsible for accepting the search request from user, searching the database and presenting the search results to user. When the user initiates a new search, database will be searched for any matching results, and the result is displayed to the user, it never searches over WWW but it searches the database only. A high level architecture of a web crawler [4] 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 from the WWW, and using some parsers 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.

III. SOFTWARE ARCHITECTURE

The architecture and model of our web crawling system is broadly decomposed into five stages.
The figure 2 depicts the flow of data from the World Wide Web to the crawler system. The user gives a URL or set of URL to the scheduler, which requests the downloader to download the page of the particular URL. The downloader, having downloaded the page, sends the page contents to the HTML parser, which filters the contents and feeds the output to the scheduler. The scheduler stores the metadata in the database. The database maintains the list of URLs from the particular page in the queue. When the user request for search, by providing a keyword, it's fed to the searching agent, which uses the information in the storage to give the final output.

1. HTML parser

We have designed a HTML parser that will scan the web pages and fetch interesting items such as title, content and link. Other functionalities such as discarding unnecessary items and restoring relative hyperlink (part name link) to absolute hyperlink (full path link) are also to be taken care of by the HTML parser. During parsing, URLs are detected and added to a list passed to the downloader program. At this point exact duplicates are detected based on page contents and links from pages found to be duplicates are ignored to preserve bandwidth. The parser does not remove all HTML tags. It cleans superfluous tags and leaves only document structure. Information about colors, backgrounds, fonts are discarded. The resulting file sizes are typically 30% of the original size and retain most of the information needed for indexing.

2. Creating an efficient multiple HTTP connection

Multiple concurrent HTTP connection is considered to improve crawler performance. Each HTTP connection is independent of the other so that the connection can be used to download a page. A downloader is a high performance asynchronous HTTP client capable of downloading hundreds of web pages in parallel. We use multi-thread and asynchronous downloader. We use the asynchronous downloader when there is no congestion in the traffic and is used mainly in the Internet-enabled application and activeX controls to provide a responsive user-interface during file transfers. We have created multiple asynchronous downloaders, wherein each downloader works in parallel and downloads a page. The scheduler has been programmed to use multiple threads when the number of downloader object exceeds a count of 20 (in our experiment).

3. Scheduling algorithm

As we are using multiple downloaders, we propose a scheduling algorithm to use them in an efficient way. The design of the downloader scheduler algorithm is crucial as too many objects will exhaust many resources and make the system slow, too small number of downloader will degrade the system performance. The scheduler algorithm is as follows:

1. System allocates a pre-defined number of downloader objects (20 in our experiment).
2. User input a new URL to start crawler.
3. If any downloader is busy and there are new URLs to be processed, then a check is made to see if any downloader object is free. If true assign new URL to it and set its status as busy; else go to 6.
4. After the downloader object downloads the contents of web pages set its status as free.
5. If any downloader object runs longer than an upper time limit, abort it. Set its status as free.
6. If there are more than predefined number of downloader (20 in our experiment) or if all the downloader objects are busy then allocate new threads and distribute the downloader to them.
7. Continue allocating the new threads and free threads to the downloader until the number of downloader becomes less than the threshold value, provided the number of threads being used be kept under a limit.
8. Goto 3.

4. Storing the web page information in a database

After the downloader retrieves the web page information from the internet, the information is stored in a database. The database is used to maintain web page information to index the web pages so that this database can be searched, for any search keyword, as in a search engine.

5. Keyword search

A search keyword is taken from the user as input and the keyword search module searches the keyword from the database and gives the indexing result 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.

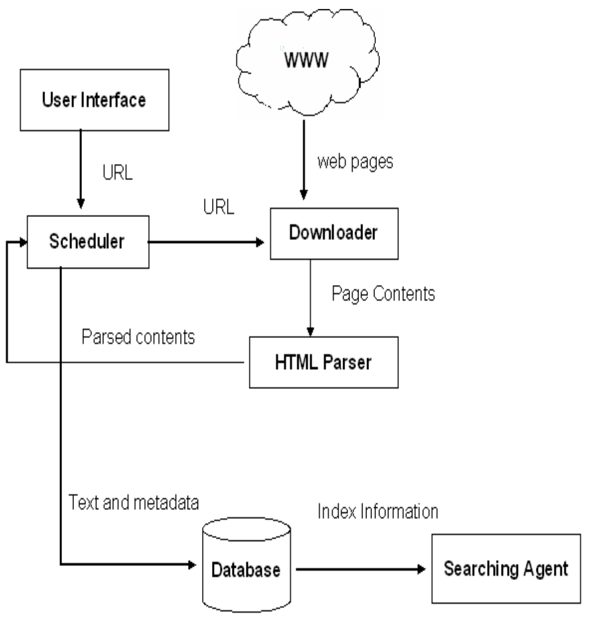

Figure 2: Software Architecture

```
Input: Start URL.  ; say u.
1.  Q = {u}. { assign the start URL to visit}
2.  while not empty Q  do
3.  Dequeue u ∈ Q
4.  Fetch the contents of the URL asynchronously.
5.  I = I ∪ {u } {Assign an index to the page visited and
    pages indexed are considered as visited}
6.  Parse the HTML web page downloaded for text and
    other links present. {u1, u2, u3, ...}
7.    for each {u1, u2, u3, …} ϵ u do
8.      if u1 ∉ I and u1 ∉ Q then
9.          Q = Q ∪ {u1}
10.     end if
11.   end for
12. end while
```

Figure 3: Web crawler algorithm.

IV. IMPLEMENTATION

This Web crawler application builds on the above mentioned modules and uses ideas from previous crawlers. This is developed in C++ works on Windows XP operating system. It makes use of Windows API, Graphics Device Interface, ActiveX controls. For database connectivity we use ODBC interface. The currently proposed web crawler uses breadth first search crawling to search the links. The proposed web crawler is deployed on a client machine. User enters the URL for example http:// rediffmail.com in the browser created. Once the start button is pressed, an automated browsing process is initiated. The HTML page contents of rediffmail.com homepage are given to the parser. The parser puts it in a suitable format as described above and the list of URLs in the HTML page are listed and stored in the frontier. The URLs are picked up from the frontier and each URL is assigned to a downloader. The status of downloader whether busy or free can be known. After the page is downloaded it is added to the database and then the particular downloader is set as free (i.e. released). We have considered 20 downloader objects, at any point of time if all downloader objects are busy threads are initiated to take up the task of the downloader. The user has a choice to stop the search process at any time if the desired results are found. The implementation details are given in table 1.

Table 1: Functionality of the web crawler application on client machine.

Feature	Support
Search for a search string	Yes
Help manual	No
Integration with other applications	No
Specifying case sensitivity for a search string	Yes
Specifying start URL	Yes
Support for Breadth First crawling	Yes
Support for Depth First crawling Support for	No
Broken link crawling Support for Archive	No
crawling	No
Check for Validity of URL specified	Yes

V. Conclusion

Web Crawler forms the back-bone of applications that facilitate Web Information Retrieval. 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. It combines the simplicity of asynchronous downloader and the advantage of using multiple threads. It reduces the consumption of resources as it is not implemented on the mainframe servers as other crawlers also reducing server management. The proposed architecture uses the available resources efficiently to make up the task done by high cost mainframe servers.

A major open issue for future work is a detailed study of how the system could become even more distributed, retaining though quality of the content of the crawled pages. Due to 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 JAVA scripts or VB scripts and also provision to support file formats like XML, RTF, PDF, Microsoft word and Microsoft PPT can be done.

VI. References

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

[2] “Effective Web Crawling” by Carlos Castillo, Department of Computer Science, University of Chile. Nov 2004.

[3] “A web crawler” by Xiaoming Liu & Dun Tan, 2000.

[4] “Web Crawling” by Baden Hughes, Department of Computer Science and Software Engineering, UniversityofMelbourne. (www.csse.unumelb.edu.au).

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

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

[7] Myspiders: Evolve Your Own Intelligent Web Crawlers Gautam Pant and Filippo Menczer, The university of Iowa City.

[8] Mercator: A scalable, Extensible Web Crawler, Allan Heydon and Marc Najork.