IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 11, May 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

Faster and Efficient Web Crawling with Parallel Migrating Web Crawler

Akansha Singh[1], Krishna Kant Singh[2]

[1]Deptt. Of Information Technology , AKGEC

Ghaziabad, India

[2]Deptt. Of Electronics & Communication

AKGEC

Ghaziabad, India

Abstract

A Web crawler is a module of a search engine that fetches data from various servers. Web crawlers are an essential component to search engines; running a web crawler is a challenging task. It is a time-taking process to gather data from various sources around the world. Such a single process faces limitations on the processing power of a single machine and one network connection. This module demands much processing power and network consumption. This paper aims at designing and implementing such a parallel migrating crawler in which the work of a crawler is divided amongst a number of independent and parallel crawlers which migrate to different machines to improve network efficiency and speed up the downloading. The migration and parallel working of the proposed design was experimented and the results were recorded.

Keywords: *web crawler, url ,Migrating*

1. Introduction

The World Wide Web is a system of interlinked hypertext documents accessed via the Internet. English physicist Tim Berners-Lee, now the Director of the World Wide Web Consortium, wrote a proposal in March 1989 for what would eventually become the World Wide Web [1]. With the ever expanding Internet, it is difficult to keep track of information added by new sites and new pages being uploaded or changed everyday. While the Internet is nearing chaos, it is difficult for a user to find correct information in a timely manner. Today's search engines are greatly used to get whereabouts of relevant information very quickly. They are like maps and signs which point the user in right direction. A search engine consists of following modules:

- A crawling module which fetches pages from Web servers known as web crawler.
- Indexing and analysis modules which extract information from the fetched pages and organize the information
- A front-end user interface and a supporting querying engine which queries the database and presents the results of searches.

2. Web Crawler

Web crawlers are a part of the search engines that fetch pages from the Web and extract information [3]. A simple crawler algorithm is as follows:

Crawler ( )

1. *Do Forever*
2. *Begin*
3. *Read a URL from the set of seed URL's*
4. *Determine the IP-address for the Host name*
5. *Download the Robot.txt file, which carries download information and also includes the files to be excluded by the crawler*
6. *Determine the protocol of underlying Host like HTTP, FTP, GOPHER*
7. *Based on this protocol, download the document*
8. *Check whether the document has already been downloaded or not*
9. *If the document is a fresh one,*
10. *Then*
11. *store it and extract the links and references to other sides from that document*
12. *Else*
13. *Abandon the document*
14. *End*

The Web crawler is given 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. [2]

3. Related Work

The literature survey shows that a number of modifications in the basic crawler have been done to improve the crawling

speed.**PARALLEL CRAWLERS [6],** A crawler can either be centrally managed or totally distributed. The authors mention that distributed crawlers are advantageous than multithreaded crawlers or standalone crawlers on the counts of scalability, efficiency and throughput. If network dispersion and network load reduction are done, parallel crawlers can yield good results. Their system utilizes memory of the machines and there is no disk access.

PARCAHYD [2] In this, work it has been proposed that if the links contained within a document become available to the crawler before an instance of crawler starts downloading the documents itself, then downloading of its linked documents can be carried out in parallel by other instances of the crawler. Therefore, it is proposed that meta-information in the form Table Of Links (TOL) consisting of the links contained in a document be provided and stored external to the document in the form of a file with the same name as document but with different extension . This one time extraction of TOL can be done at the time of creation of the document

MIGRATING CRAWLER [4], an alternative approach to Web crawling is based on *mobile* crawlers. The authors propose that the crawlers are transferred to the source(s) where the data resides in order to filter out any unwanted data locally before transferring it back to the search engine. This reduces network load and speeds up the indexing phase inside the search engine.

MERCATOR [7] is a scalable and extensible crawler, now rolled into the Altavista search engine. The authors of [7] discuss implementation issues to be acknowledged for developing a parallel crawler like traps and bottlenecks, which can deteriorate performance. They discuss pros and cons of different coordination modes and evaluation criteria. In brief, they concur that the communication overhead does not increase linearly as more crawlers are added, throughput of the system increases linearly as more nodes are added and the *quality* of the system, i.e. the ability to get "important" pages first, does not decrease with increase in the number of crawler processes.

4. The Problem

With the help of a study done on the various crawling methods several problems were identified, these are as follows

- Due to the rapid growth of the Web the downloading creates a bottleneck at the downloader side.
- Further the pages may be downloaded in duplicates that generates unnecessary network load.
- In addition, Kahle [9] reports that the average online time of a page is only 75 days, which leads to an update rate of 600GB per month that should be downloaded to keep collection up to date. So, the crawler should revisit the already downloaded pages to check the updation.
- Thus, in order to keep the database of a search engine up to date, crawlers must constantly retrieve/update Web pages as fast as possible.

5. Proposed Solution

The proposed solution to the above problems is to decentralize and perform site-based distribution of work among the machines and simultaneously crawl as many domains as possible. This paper aims at designing a centrally managed migrating parallel crawler to crawl websites. The crawling function is logically migrated to different machines which send back the filtered and compressed data to the central machine which saves time and bandwidth. The architecture proposed in this paper is shown in figure 1. The major focus is on the design proposed for the crawler system, which implements the idea of parallel migrating crawler. The crawler system itself consists of several specialized components, in particular a *central crawler*, one or more crawl frontiers, and a local database of each crawl frontier. This data is transferred to the central crawler after compression and filtering which reduces the network bandwidth overhead. These crawl frontiers, are logically migrated on different machines to increase the system performance. The central crawler is responsible for receiving the URL input stream from the applications and forwarding it to the available crawl frontiers. The crawler system is composed of a central crawler and a number of crawl frontiers which perform the task of crawling.

1. Central Crawler: The central crawler is the central component of the system, and the first component that is started up. Afterwards, other components are started and register with the central crawler to offer or request services. It is like the server in Client server. It is the only component visible to the other components, as the crawl frontiers work independently. In general, the goal of the central crawler is to download pages in approximately the order specified by the application, while reordering requests as needed to maintain high performance without putting too much load on any particular web server. The central crawler has a list of URLs to crawl. The URLs are sent in batches to the crawl frontiers, making sure that a certain interval between requests to the same server is observed. The central crawler has a seed URL list which is to be crawled unlike conventional crawling the central crawler has a set of available crawl frontiers which have registered themselves with the central crawler which are logically migrated to different locations. The central crawler assigns different URLs to all the crawl frontiers and in turn the crawl frontiers begin to crawl the received URL. As the crawl frontier completes its crawling, the central crawler receives the compressed downloaded content from the crawl frontiers. The central crawler is implemented using java RMI .The Algorithm is as follows:

Central Crawler ()

1. *Do Forever*

2. Begin

3. Register crawl frontiers

4. Read a URL from the set of seed URL's

5. Determine the IP-address for the Host n€ame

6. Download the Robot.txt file, which carries download information and also includes the files to be excluded by the crawler.

7. while seed URL list is not empty or crawl frontiers are available

8. Dispatch URL to available crawl frontier.

9. Wait for result from crawl frontiers.

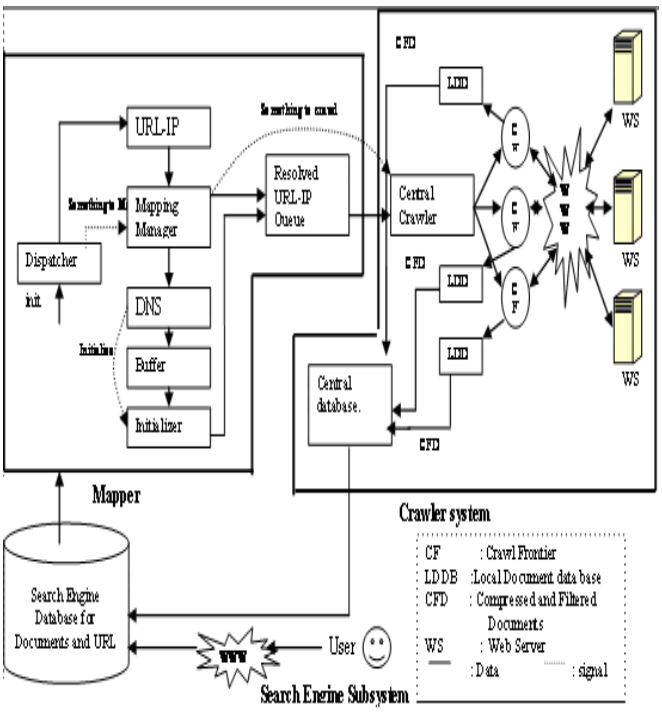

Figure 1: Proposed Architecture

10. receive downloaded content from crawl frontiers.

11. store the local data of each crawl frontier in the central database.

12. End

2. Crawl Frontiers (CF): The crawl frontier component, implemented in JAVA, performs the basic function of a breadth first crawler, i.e., it fetches files from the web by opening up connections to different servers. It is like the client in client /server architecture The files are written in the local document database available with each crawl frontier. The application then parses each downloaded page for hyperlinks, checks whether these URLs have already been encountered before, and if not, adds them to the queue containing the links to be visited. The crawl frontier also shows the time taken by it a crawl a particular url , i.e., the url and the hyperlinks found on visiting that url. In the implementation there are two queues one which contains the internal links of the website and the other contains the external links encountered. This is done to set a priority that first all the internal links are visited and then the external links are visited. The downloaded files are then compressed using java classes and transferred to the central database. The crawl frontiers is also implemented in java. The algorithm is as follows:

Crawl frontier ()

1. Do Forever

2. Begin

3. Register with central server by sending the IP address of the machine.

4. Wait for URL to crawl

5. Receive URL from central crawler.

6. Add this URL to the list of URLs to visit.

7. while URL list is not empty

8. remove the first URL from the list

9. visit the URL.

10. save the page in the local database.

11. parse the page and find the URL hyper links in the page

12. if the links are not present in the to visit list

13. add the URL to the to_visit list

14. Compress the downloaded content.

15. Send the compressed content to the central crawler crawler.

16. End

3. Local Document Database (LDDB): When the crawl frontier runs it requires some memory space to save the downloaded content for this purpose each crawl frontier has its own local database which is known as the local document database. The crawl frontiers save the downloaded content in a directory in this database .It is the storage area of the machine on which the crawl frontier is running.

4. Central Database: As in conventional crawlers this database is the one that communicates with the database of the search engine. The central database stores the list of URLs received from the application and it also stores the final downloaded documents which is composed of the various documents downloaded by the different crawl frontiers.

5. Web Server (WS): The term web server means a computer program that is responsible for accepting HTTP requests from *clients* user agents such as web browsers, and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects (images, etc.).

The **mapper system** is an existing system, which works between the search engine and the crawler to map the request and response between the search engine and the crawler. The Mapper system is discussed in detail in [3].The major components of this system are URL-IP Queue consisting of a queue of unique seed URL-IP pairs. Resolved URL queue that stores URLs which have been resolved for their IP addresses and acts as an input to the Crawl Manager. URL Dispatcher it sends a signal: Something to Map to the Mapping manager. Mapping Manager creates multiple worker threads called URL Mapper. It extracts URL-IP pairs from the URL-IP Queue and assembles a set of such pairs called URL-IP set. Each URL- Mapper is given a set for mapping.This component gets a URL-IP set as input from the Mapping Manager. It examines each URL-IP pair and if IP is blank then the URL is sent to the DNS Resolver. After the URL has been resolved for its IP, it is stored in the Resolved URL Queue. It sends a signal Something to crawl to the Crawl Manager.

6. Implementation Details

A number of modules were implemented in java which were integrated together to form the entire parallel migrating crawler. These modules were as follows:

1. CServerInterface: This is the first module implemented , this is the remote interface that is used to implement java RMI. This interface declares the remote methods.As a member of a remote interface, the showclient , transfer and getDownloadURLS methods are remote method.

2. CMainServer : The remote methods declared in the above interface are defined in this part. These are the methods that are used between the central crawler and the crawl frontier.

3. CServer: This is the central crawler that is to be started first, it registers with itself the crawl frontiers and dispatches URLs to them. It is the RMI server.

4. CBasicCrawler1: This is the crawl frontier code that is used to perform the basic crawling process. It is the RMI client. It receives URLs from the central crawler and run on different machines to collect pages. It also contains method to compress and filter the data.

The experiment was carried out on 3 machines. That is there were three crawl frontiers and one central server in the experiment. The crawl limit was kept as 150 i.e., the crawl frontiers crawled up to 150 links with a high speed internet of around 236.8 Kbps. Data was collected with the above mentioned set up in terms of the downloaded content , time that each machine took in downloading a particular site .Also it was seen that after compression a large amount of bandwidth was saved since the volume of downloaded content was reduced greatly.

7. Results

The results obtained are summarized in Table 1.

TABLE 1

CF	URL	CL	C D	AC	T
1	U1	150	2. 84	73 4	2137
2	U2	150	3.87	959	2491
3	U3	150	3.16	923	2315
Total			9.87	2616	2491

Where

CF: Crawl Frontier

U1: http://www.coe-roorkee.com

U2: http://www.freshersworld.com

U3: http://www.akgec.org

CL: Crawl limit

CD: Content downloaded in MB

AC: Content After compression in KB

T: Time in seconds

The total time will be the maximum of the three times as all the CFs were working in parallel. The same three urls were given to a traditional or centralized crawler with the same operating conditions and the results were quite interesting the same amount of content was downloaded in around 6950 seconds which is slightly greater than the sum of the time taken by the three crawl frontiers individually whereas in our crawler will take a maximum of 2491 seconds to download the same amount of content. Also since our crawler has an inbuilt feature of compression the amount of data that has to be sent over the network is also greatly reduced saving the bandwidth. The above results show that a migrating parallel crawler in which the work of a crawler is divided amongst a number of independent parallel

crawlers called crawl frontiers improve network efficiency and speed up the downloading.

8.Conclusion

Crawlers are being used more and more often to collect Web data for search engine, caches, and data mining. As the size of the Web grows, it becomes increasingly important to use parallel crawlers. Unfortunately, very little is known (at least in the open literature) about options for parallelizing and migrating crawlers and their performance. This paper addresses this shortcoming by presenting a parallel migrating architecture, and by studying its performance. The authors believe that this paper offers some useful guidelines for crawler designers, helping them select the right number of crawling processes, or select the proper inter-process coordination scheme. In summary, the main conclusions of my design, implementation and the results obtained were the following:

- Decentralizing the crawling process is a good solution for catering the needs of the ever increasing size of web.
- When a number of crawling processes migrate to different locations and run parallel they make the crawling process fast and they save enormous amount of time in crawling.
- The documents collected at each site are filtered. So only the relevant pages are sent back to the central crawler and this saves network bandwidth.
- The documents before being sent to the central crawler are compressed locally and then sent to the central crawler which saves a large amount network bandwidth.

References

[1] Douglas E. Comer, "*The Internet Book*", Prentice Hall of India, New Delhi, 2001.

[2] Monica Peshave, "How Search Engines Work And A Web Crawler Application"

[3] A.K. Sharma, J.P. Gupta, D. P. Aggarwal, "*PARCAHYDE*: *An Architecture of a Parallel Crawler based on Augmented Hypertext Documents.*"

[4] Joachim Hammer,Jan Fiedler **"**Using Mobile Crawlers to Search the Web Efficiently"

[5] V. Shkapenyuk , T. Suel, "*Design and implementation of a high performance distributed Web crawler*". In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), San Jose, CA Feb. 26--March 1, pages 357 -368, 2002,

[6] J. Cho and H.Garcia-Molina, "*Parallel crawlers*". In Proceedings of the Eleventh International World Wide Web Conference, 2002, pp. 124 - 135,

[7] A. Heydon and M. Najork, "*Mercator: A scalable, extensible web crawle*r".World Wide Web, vol. 2, no. 4, pp. 219 -229, 1999.

[8] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "*Ubicrawler: A scalablefully distributed web crawler*". In Proceedings of AusWeb02 - The Eighth Australian World Wide Web Conference, Queensland, Australia, 2002,

[9] B. Kahle. Achieving the Internet. *Scientific American*, 1996.