Design of a Priority Based Frequency Regulated Incremental Crawler

Niraj Singhal
School of Computer Engineering & Information Technology,
Shobhit University,
Meerut, Uttar Pradesh, India

Ashutosh Dixit
Computer Engineering Department
YMCA Institute of Engineering, Faridabad,
Haryana, India

Dr. A. K. Sharma
Computer Engineering Department
YMCA Institute of Engineering, Faridabad,
Haryana, India

ABSTRACT
The World Wide Web is a huge source of hyperlinked information contained in hypertext documents. Search engines use web crawlers to collect these documents from web for the purpose of storage and indexing. However, many of these documents contain dynamic information which gets changed on daily, weekly, monthly or yearly basis and hence we need to refresh the search engine side storage so that latest information is made available to the user. An incremental crawler visits the web repeatedly after a specific interval for updating its collection. In this paper to regulate the revisiting frequency a novel mechanism and a novel architecture for incremental crawler is being proposed.

Keywords : web search engine, incremental crawler, frequency, regulated, priority, crawl workers.

1. INTRODUCTION

Internet [6], an interconnection of millions of computers around the world, is a global information system that is logically linked together by a globally unique address space based on the Internet Protocol. WWW [1,2,5,7,16,17,18] is a web of hyperlinked repository of trillions of hypertext documents [19] lying on different websites, distributed over far and distant geographical locations. In fact, the size of web is so enormous that it is frustrating and tedious task to search the right information at the right time.

Web search engines [2,4] employ crawlers to continuously collect web pages from the web. The downloaded pages are indexed and stored in a database as shown in Figure 1. This continuous updation of database renders a search engine more reliable source of right and updated information [7,9].

It may be noted (see Figure 1) that the crawler is provided with a list of URLs. It visits the destined websites and depending upon the host protocol, downloads the desired documents.

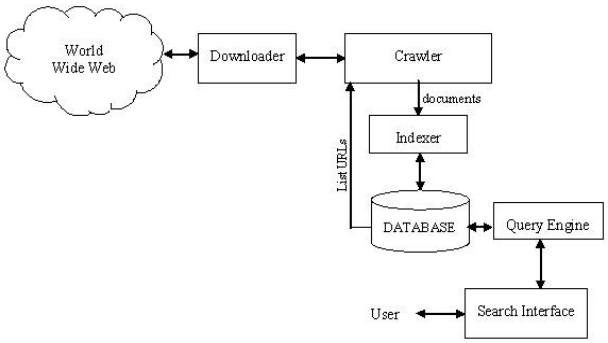

Figure 1. Architecture of a typical web search engine

The general algorithm of a web crawler is given below :

Begin
Read a URL from the set of seed URLs;
Determine the IP address for the host name;
Download *Robot.txt* file which carries downloading permissions and also specifies the files to be excluded by the crawler;
Determine protocol of underlying host like http, ftp, gopher etc.;
Based on the protocol of the host, download the document;
Identify the document format like doc, html, or pdf etc.;
Check whether the document has already been downloaded or not;
If the document is fresh one
then
Read it and extract the links or references to the other cites from that documents;
else
Continue;
Convert the URL links into their absolute URL equivalents;
Add the URLs to set of seed URLs;
End.

2. RELATED WORK

A *periodic crawler* [12] visits the web sites until its collection has a desirable number of pages and stop visiting pages. Whenever it needs to refresh its collection, it revisits the sites, creates a new collection and replaces the old collection with the new. Whereas an *incremental crawler* [12] refresh existing pages and replaces *less important existing pages* with *more important new pages*. It crawls (Figure 2) the web sites continuously, refreshes local collection and provides fresh information to the user. Good freshness can only be guaranteed significantly by simply revisiting all pages very frequently and selecting the page that will increase the freshness most significantly.

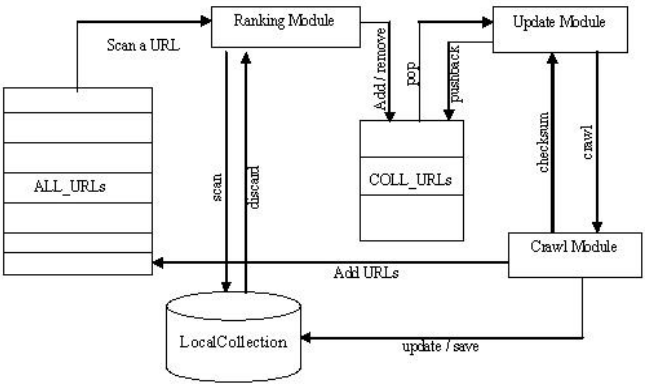

Figure 2. Architecture of an Incremental web crawler

The incremental crawler consists of three main data structures *ALL_URLs*, *COLL_URLs* and *LocalCollection*, and three main software modules *RankingModule*, *UpdateModule* and *CrawlModule*.

Where :

ALL_URLs	:	Set of All URLs accessed/to be accessed.
COLL_URLs	:	Set of All URLs in the Local Collection
LocalCollection	:	Collection of downloaded pages related to URLs in COLL_URLs

The RankingModule continuously scans through ALL_URLs and the LocalCollection to take the right refinement decision. UpdateModule selects a URL from COLL_URLs, downloads the page, and if changed; updates the LocalCollection. The CrawlModule crawls a page and saves/updates the page in the LocalCollection based on the request from UpdateModule. The CrawlModule also extracts all links/ URLs on the crawled page and adds in ALL_URLs.

The design of an incremental crawler needs to address the following issues :-

(a). Keep the local collection fresh : The freshness of pages in local collection depends on the strategy used, so the crawler should use the best policies to keep the local collection fresh. This includes adjusting the revisit frequency for a page based on its estimated change frequency.

(b). Improve quality of the local collection : The crawler should increase quality of the local collection by replacing *less important pages* with *more important pages*. It is necessary because of two reasons; firstly, pages are constantly created and removed, and some of the pages created may be more important than existing pages in the local collection. So, the crawler needs to replace *less important existing pages* with *more important new pages*. Secondly, the importance of existing pages also changes over time. So, when some existing pages become less important than previously ignored pages, then the crawler should replace *less important existing pages* with *previously ignored new pages*.

Some approaches [3,5,12] have proposed many ways to manage dynamic information on the web. When pages are changing very fast, then these crawlers need to visit the pages as frequently as possible. Today when web size has become very large; these revisits not only engage the network traffic for a longer time but the crawler will also not be able to crawl the complete web in feasible time.

Sharma et al [2] has addressed the above issues by managing the dynamic information more efficiently by putting volatile information using separate HTML tags. These tags and information are stored in a separate file having same name with extension .TVI (table of variable information). The TVI file is updated every time the changes are made to the hypertext document. The crawler checks the contents of TVI file for any changes and accordingly updates its collection.

A critical look at the available literature [5,7,10,12,13] indicates that for the purpose of revisiting policies to refresh a web page; almost all studies have focussed on frequency of change of a page as a parameter. Since all pages do not change at same interval of time, they can not be refreshed at same frequency. Rather, the pages that change fast need to be refreshed fast as compared to those change least.

In this paper, an alternate approach has been proposed to manage the process of revisiting of a web site. It employs an ecology of crawl workers to crawl the web sites. Crawl manager extracts URLs from each queue of URLs and distribute them among crawl workers.

3. PROPOSED WORK

Based upon updation activity, web documents can be categorized and grouped as follows :

(i). The pages that changes very frequently (say every six hours)

(ii). The pages that changes frequently (say every twelve hours)

(iii). The pages that changes less frequently (say daily)

(iv). The pages that changes further less frequently (say weekly)

(v). The pages that changes lesser frequently (say fortnightly)

(vi). The pages that changes further lesser frequently (say monthly)

(vii). The pages that changes least frequently(others)

Keeping in view the above categorization, the crawler may visit a site frequently and the frequency of visits may be adjusted according to the category of the site. Frequency of updation is a major factor that decides as when to revisit a page, and is directly proportional to the number of visits to the site and hence the network traffic. So, a page that change very frequently say every six hours, needs to be revisited frequently as compared to the page that change less frequently say every year, needs to be revisited least frequently.

The complete architecture of the proposed priority based frequency regulated incremental crawling module is given in Figure 3.

The crawler maintains following main data structures :

1. UrlQ(1…n) :

   Store URLs to be crawled category wise in 'n' queues and within each queue URLs arearranged PageRank wise. Initially all Url queues is populated with seed URLs.

2. Documents/URLs buffer :

   Buffer to store URLs and web documents crawled by the Crawl workers.

3. Database :

   Stores web pages (*DbPages)*, related URLs (*DbURLs)* and new URLs found (*DbNewURLs)* in the Database.

The main software modules of the proposed crawler are *CrawlManager, PriorityManager, Save/Update,* an update trigger (*Upd_Trig), and Crawl Workers (CW1..CWn).*

1. CrawlManager :

   It extracts URLs from URL queues and distribute them among the various Crawl workers *CW1…CWn*.
   Algorithm for Crawl manager is given in Figure 4.

2. Save/Update module :

   Updates the Database in search/insert fashion by replacing old version with fresh version of the page. If such page does not exist, insert the new page.
   Algorithm for Save/Update module is given in Figure 5.

3. Upd_Trig :

   An update trigger that gets activated after regular intervals and supplies all URLs from the Database to the *PriorityManger*.

4. PriorityManager :

   Computes refresh rate/frequency of the URLs dynamically and populates URL queues *UrlQ(1..n)* for next refresh cycles. It also prompts the CrawlManager to proceed further.
   Algorithm for Priority manager is given in Figure 6.

5. Crawl Workers :
   Crawl Workers crawl the web, download pages and extract new URLs embedded in the pages.

The refresh rate/frequency is computed [5] dynamically by the following formula :

$t_{n+1} = t_n + \Delta t$	(1)

Where :

t_n is current refresh time for any site

t_{n+1} is adjusted refresh time and

Δt is change in refresh time to be calculated dynamically

The value of Δt may be positive or negative, based upon the degree of success (p_c) that the site contains the volatile documents. The degree of success is computed in terms of number of hits by detecting the frequency of changes occurred in the documents on a site.

A unit step function $u(p_c)$ for the computation of Δt is,

$\Delta t=\{(1-p_c/p_g)*u\ (p_c-p_g)+(1-p_c/p_l)*u\ (p_l-p_c)\}* t_n$	(2)

Where p_g, p_l are the boundary conditions (upper and lower threshold values of p_c respectively)

$$\text{And } u(p_c) = \begin{cases} 1 & \text{if } p_c > 0 \\ 0 & \text{otherwise} \end{cases}$$

The URL queues UrlQ(n) store URLs refreshing time wise wherein they are stored PageRank wise. All the URLs to be crawled may be distributed among UrlQ(1..n) based upon formula 3.

For each UrlQ(n) = $(n-1)^2$ *20 units to n^2 * 20 Units	(3)

i.e. range of refresh time limits for various queues are as follows :

UrlQ(1) : 0 to < 20 Units
UrlQ(2) : 20 to < 40 Units
UrlQ(3) : 40 to <180 Units
……
UrlQ(9) : 1280 to <1620 Units
UrlQ(10): >=1620 Units

The frequency of revisiting may be decided as per formula (4) for each n^{th} queue of URLs i.e. frequency of selection of URL to be crawled from each of the queues are as follows :

For each UrlQ(n) $n^2 * 20\ \mu$ sec	(4)

UrlQ(1) : at each 20 μ Sec
UrlQ(2) : at each 80 μ Sec
UrlQ(3) : at each 180 μ Sec
……
UrlQ(10) : at each 2000 μ Sec

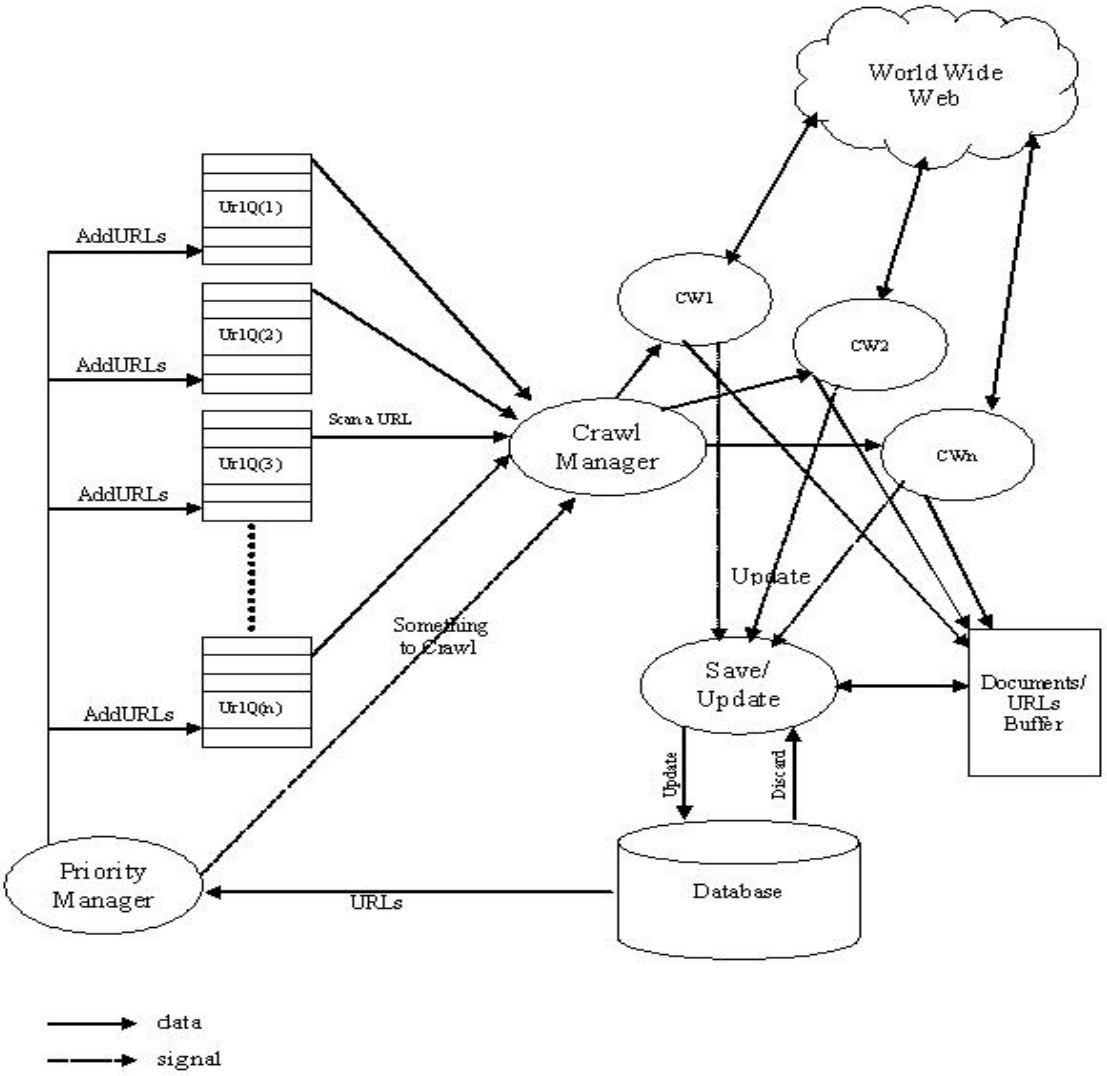

Figure 3. Architecture of proposed crawling module

```
Crawl Manager()
{
   Read a URL from UrlQ(n)
   Creates a crawl worker CW(n)
   Download new page from the web and extracts new URLs found
in the page
   If  URL exists in buffer
   then
      If page changed, updates it in Document/URL buffer
   else
      add URL, page to Document/URL buffer
   Add/replace new URLs to Document/URL buffer
   Passes an Update signal to SaveUpdate()
}
```

Figure 4. Algorithm for Crawl Manager

```
SaveUpdate()
{
   Reads a URL & page downloaded from buffer
   If URL exists in database
   then
      If page changed then updates it in the database
   else
      Find Page Rank of the page
      Select page_to_discard page with lowest page rank
      Discard  page_to_discard and its URL
      Adds URL & Page in the database
      Pass all new URLs found in new page to the Priority Manager
}
```

Figure 5. Algorithm for SaveUpdate

```
Priority Manager()
{
   While (URLs)       // accepted URLs from the database
   {
      Find Page Rank, Frequency of refreshing of a URL
      Add/replace URL in appropriate queue of URLs UrlQ(n), refreshing time wise, wherein arranged page rank wise
      Passes Something to crawl signal to the Crawl Manager
   }
}
```

Figure 6. Algorithm for Priority Manager

The CrawlManager selects the next URLs *url_to_extract* from each queue of URLs UrlQ(1..n) and distributes them among various crawl workers *CW1*.....*CWn*. Each Crawl Worker crawls the web, downloads page and extracts new URLs embedded in the page. The extracted URLs, downloaded documents and new URLs are stored in the Document/URL buffer and signal called "*Update"* is send to *Save/Update* process.

Save/Update picks-up extracted URLs, downloaded documents and new URLs stored in the Document/URL buffer and updates the Database in search/insert fashion. It checks for existence of URL extracted in *DbURLs* of Database, if URL exists; replaces old version *page_existing* with new version *page_extracted* of the page. If such URL does not exist, it inserts the new URL and *page_extracted* to the Database. It also adds all the newURLs found on page from Buffer to the *DbNewURLs*.

An update trigger *Upd_Trig* in the Database, gets activated and the Priority Manager extracts URLs from the Database, computes refresh rate/frequency of the URLs dynamically; and adds them to the respective queues of URLs *UrlQ(1…n)* and populates them. It also sends a signal "*something_to_crawl"* to the Crawl Manager for further processing.

In fact, the Crawl Manager extracts URLs from each URL queue and distribute them among several crawl workers. Each crawl worker crawls the web and updates the Database. It gives proper chance to each URL to be crawled. The priority manager computes page refresh frequency dynamically to store all URLs in specific UrlQ(1…n).

Thus, the architecture designed in this work, is not only incremental but also scalable that can be parallelized at URL queues and crawl workers level, responsible for downloading documents from the web.

4. CONCLUSION

The proposed architecture of incremental web crawler manages the process of revisiting of a web site with a view to maintain fairly fresh documents at the search engine site. The computation of refresh time helps in improving the effectiveness of the crawling system by efficiently managing the revisiting frequency of a website; and appropriate chance to each type of website to be crawled at a fast rate. Moreover, the architecture is suitable for parallel application.

5. REFERENCES

[1] A. K. Sharma, J. P. Gupta, D. P. Agarwal, "Augment Hypertext Documents suitable for parallel crawlers", accepted for presentation and inclusion in the proceedings of WITSA-2003, a National workshop on Information Technology Services and Applications, Feb'2003, New Delhi.

[2] A. K. Sharma, J. P. Gupta, D. P. Agarwal, " A novel approach towards management of Volatile Information" Journal of CSI, Vol. 33 No. 1, pp 18-27, Sept' 2003.

[3] Alexandros Ntoulas, Junghoo Cho, Christopher Olston, "What's new on the Web ? The Evolution of the Web from a Search Engine perspective.", In Proceedings of the World-Wide Web Conference (WWW), May 2004.

[4] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, Sriram Raghavan "Searching the Web." ACM Transactions on Internet Technology, *1(1): August 2001*

[5] Ashutosh Dixit, Harish Kumar and A.K Sharma, "Self Adjusting Refresh Time Based Architecture For Incremental Web Crawler", International Journal of Computer Science and Network Security (IJCSNS), Vol 8, No12, Dec 2008.

[6] *Barry M. Leiner*, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard, Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, Stephen Wolff, *"*A Brief History of the Internet", www.isoc.org/internet/history.

[7] Brian E. Brewington and George Cybenko. "How dynamic is the web.", In Proceedings of the Ninth International World-Wide Web Conference, Amsterdam, Netherlands, May 2000.

[8] C. Dyreson, H.-L. Lin, Y. Wang, "Managing Versions of Web Documents in a Transaction-time Web Server" In Proceedings of the World-Wide Web Conference.

[9] Dirk Lewandowski, "Web searching, search engines and Information Retrieval, Information Services & Use", 25 (2005) 137-147, IOS Press, 2005

[10] Heydon A., Najork M., "Mercator: A scalable, extensible Web crawler.", World Wide Web, vol. 2, no. 4, pp. 219-229, 1999.

[11] J. Dean and M. Henzinger, "Finding related pages in the world wide web", Proceedings of the 8th International World Wide Web Conference (WWW8), pages 1467-1479, 1999.

[12] Junghoo Cho and Hector Garcia-Molina. 2000a. "The evolution of the web and implications for an incremental crawler"., In Proceedings of the 26th International Conference on Very Large Databases.

[13] Junghoo Cho and Hector Garcia-Molina, "Estimating frequency of change", 2000, Submitted to VLDB 2000, Research track.

[14] Komal Kumar Bhatia, A. K. Sharma, "A Framework for Domain-Specific Interface Mapper (DSIM)", International Journal of Computer Science and Network Security (IJCSNS), Vol 8, No12, Dec 2008.

[15] Mark Najork, Allan Heydon, "High- Performance Web Crawling", September 2001

[16] Mike, Burner, "Crawling towards Eternity : Building an archive of the World Wide Web", Web Techniques Magazine, 2(5), May 1997

[17] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hyper textual Web search engine". Proceedings of the Seventh International World Wide Web Conference, pages 107—117, April 1998.

[18] S. Chakrabarti, M. van den Berg, and B. Dom, "Distributed hypertext resource discovery through examples", Proceedings of the 25th International Conference on Very Large Databases (VLDB), pages 375-386, 1999.

[19] www.w3.org/hypertext/WWW/MarkUp/MarkUp.html--official HTML specification

[20] Y. Yang, S. Slattery, and R. Ghani, "A study of approaches to hypertext categorization", Journal of Intelligent Information Systems. Kluwer Academic Press, 2001.