International Journal of Computer Applications (0975 – 8887)
Volume 86 – No 5, January 2014

Estimating Page Importance based on Page Accessing Frequency

Komal Sachdeva
Assistant Professor
Manav Rachna College of Engineering, Faridabad, India

Ashutosh Dixit, Ph.D
Associate Professor
YMCA University of Science and Technology, Faridabad, India

ABSTRACT

With the vast growth of the Internet, many web pages are available online. Search engines use a component called as web crawlers for collecting these web pages from the web for storage and indexing. Many web pages are autonomous and are updated independent of the users. .As the web pages are updated autonomously; users do not come to know of how often the sources change. An incremental crawler visits the web repeatedly after a specific interval of time for the updation of its collection. Users are benefited by knowing the page importance based upon the page accessing frequency. This paper finds out the page importance based on page accessing frequency and also architecture for the same is also proposed.

General Terms

Page Importance Based On Page Accessing Frequency.

Keywords

Search Engine, Web Crawler, Page access Frequency.

1. INTRODUCTION

The World Wide Web [2, 3, 12] is a system that contains hypertext documents which are linked to each other; world wide web is accessed by web browser from web servers. These hypertext documents can contain text, images, audio and video data. Hyperlinks allow one to move to the other information resources from the recent one and also come back. In this way information is explored and creates a rich set of world wide information.

The web is a large repository of text documents, multimedia, images and number of other information. Because on www, large no. of web pages is available, search engine is dependent upon web crawlers for gathering the required information on the web.

A web crawler [13, 14] is a computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion. A crawler starts off by placing an initial set of URLs, in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler extracts a URL, downloads the page, extracts URLs from the downloaded pages, and places the new URLs in the queue. This process is repeated and the collected pages are later used by other applications, such as a Web search engine.

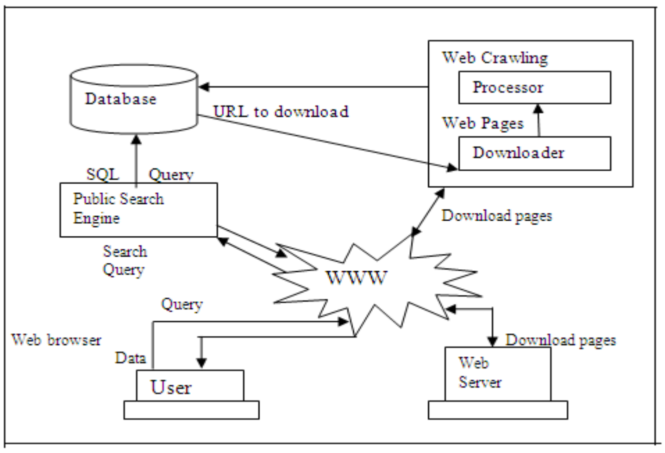


Figure1. Architecture of search engine

In Figure 1, the search engine accepts the query from the user. An interface is provided by the search engine to the user so that users submit the queries. And it contains the mechanism for serving these queries. This is the only part which is visible to the end-users.

The database stores the data crawled by the web crawlers. The search engine queries the database so as to answer any user's request. The database also feeds the downloader with the URLs to be downloaded .The processor processes the URLs it takes from the downloader and updates the database with the fresh information (URLs).

The basic algorithm for the Web Crawler is given below:

1. Read a URL from the set of seed URLs.
2. Find out the IP address for the host name.
3. Download the Robot.txt file that carries downloading permissions and also identifies the files to be excluded by the crawler.
4. Find out the protocol of underlying host like http, ftp, gopher etc.
5. Based on the protocol of the host, download the document.
6. Identify the document format like doc, html, or PDF etc.
7. Check whether the document has previously been downloaded or not.
8. If the document is fresh one
   Then Read it and extracts the links.
9. Else
   Continue.
10. Convert the URL links into their absolute IP equivalents.
11. Add the URLs to set of seed URLs.

2. RELATED WORK

The incremental crawler selectively and incrementally updates its index and/or local collection of web pages. Instead of periodically refreshing the collection in batch mode, the incremental crawlers improves the “freshness” of the collection significantly and brings in new pages in a more appropriate manner.

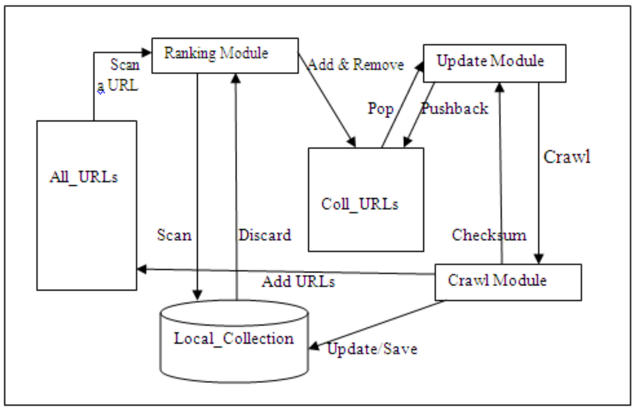


Figure2. Architecture of Incremental Crawler

Figure 2 represents the architecture of Incremental Crawler. The URLs that are accessed or to be accessed are contained by the All_URLs module. The Ranking module choses the URLs in the Coll_URLs. The Local_Collection downloads the pages related to URLs in the Coll_URLs. The Ranking Module continuously scans the All_URLs and the Local_collection and gives proper ranks to the web pages accordingly. When the Ranking Module finds that the page present in the Coll_URLs are not the updated pages, then the page in the Coll_URLs are replaced by the updated pages.

The Update Module takes the URLs from the Coll_URls, downloads the pages related to the URL, if the page is changed, then it will be updated in the Local_collection. The Crawler Module has to crawl the web page and it updates or saves the web page into the Local_collection according to the request of Update Module. It also extracts all the links (URLs) on the crawled page and adds those links in All_URLs module.

While designing the incremental crawler [2, 6] two issues must be addressed:-

a) Maintain the local collection with fresh pages:
Freshness of web pages in local collection is based on the strategy that used for it, that why the crawler should apply the best policies to maintain the local collection fresh.
In order to maintain the freshness of local collection, Revisit Frequency Calculator is used to find the appropriate revisit frequency of the crawling so that crawler can update its local collection with fresh documents.

b) Improve quality by keeping relevant pages in the local collection:
The web crawler should improve the quality of the local collection by replacing less relevant pages with more relevant pages. It is essential because pages are continuously created and destroyed, and it is also possible that some of the pages that were created may be more relevant than existing pages in the local collection. So, the crawler needs to replace less relevant existing pages with more relevant web pages.

Another reason is that, the relevancy of existing pages also changes over time. Thus, when some existing pages become less relevant than earlier ignored pages, then the web crawler should replace less relevant existing pages with earlier ignored new web pages.

Many earlier approaches like Page Rank calculate page importance through the use of the hyperlink graph of the Web. Recently, people realized that the hyperlink graph is incomplete and inaccurate as a data source for determining page importance.

In approach, [1] it proposed a way to calculate page rank in incremental crawler .According to it, the hit counter stores the number of times the page is visited. And there must be some date on which that page is added on the web.

For estimating the page importance of that page it divide the hit counter by the total number of days for that page on the web. This will evaluate the access ratio of that page for the web.

Another approach, [3] it solves the problem of searching new information from the web in incremental web search to evaluate ranking of changed web pages. For solving this problem it uses an integrated ranking framework by combining three metrics. The three metrics are Popularity Ranking, Content-based Ranking and Evolution Ranking which produce good Ranking for the changed web Pages.

3. PROPOSED WORK

There are three important characteristics of the web page that generate a scenario in which the web crawling is very difficult:

i. Large Volume of web pages.
ii. Rate of change on web pages.
iii. Finding the relevant pages on the web.

A large volume of web pages means that the web crawler can only download a fractional of the web pages and hence it is very essential that the crawler should be intelligent enough to prioritize download.

Whereas rate of change on web pages implies that web pages on the internet change very frequently, as a result, by the time the crawler is downloading the last page from a site, the web page may change or a new page has been placed/ updated on the website.

Relevant pages of the web represents the ’value’ of an individual page on the web, is a key factor for web search, Various search engine’s components like, the crawler, indexer, and ranker are usually guided by this measure. Because the scale of the web is extremely large, and the web evolves dynamically, accurately calculating the importance scores of web pages becomes critical, and also poses a great challenge to search engines.

The proposed architecture in Figure 3 tries to solve all these problems

The architecture is composed of the following Modules/Data Structure:

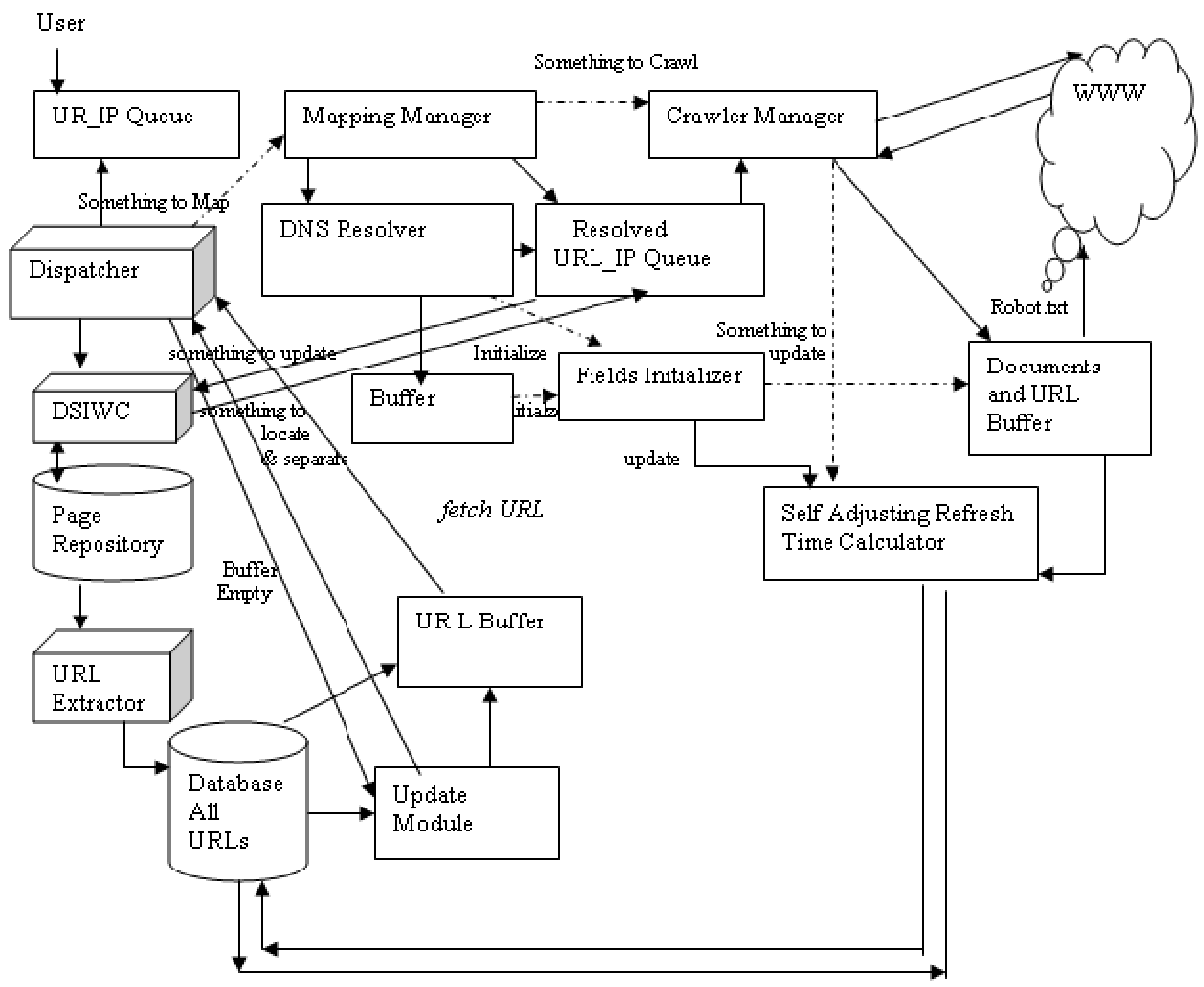


Figure 3. Block Diagram of Page Relevant Incremental Crawler

The block diagram of Page Relevant Incremental Crawler consists of the various components:

- **URL_IP Queue:** A set of seed URL-IP pairs.
- **Resolved URL_IP Queue: Set** of URLs which have been resolved for their IP addresses.
- **Data Base:** It contains a database of downloaded documents and their URL-IP pairs. Priority queue is used to store URL_IP pairs.
- **Refresh Time Calculator:** This component uses the formula for computing the next refresh time.
- **Dispatcher:** Dispatcher waits for the *fetch URL* signal and upon receiving this signal, it fetches an URL from the URL Buffer so that DSIWC in turn can download the corresponding web page. If dispatcher finds the URL Buffer empty during this operation, then it sends Buffer Empty signal to the Update Module so that it can add more URLs in the URL Buffer.
- **DNS Resolver:** The DNS revolver uses the services of internet for translating the URLs to their corresponding IP addresses and stores them into the resolved URL_IP queue. The new URLs with the IP addresses are stored in the buffer. A signal *initialize* is sent to the Field initializer.
- **Buffer:** It stores the new URLs with their corresponding IP addresses.
- **Fields Initializer:** It initializes the URL_fields in URL record for which object structure
- **Doc ID** field is the unique identifier for each document.
- **Refresh Time:** field stores the duration of the document to be refreshed. After initializing by some default value, this field is dynamically updated by refresh time calculator module.
- **LastCrawlTime:** field stores the date Time stamp of last time page was crawled.
- **Status:** field represent whether URL is present in URL-IP queue or not.

If Status is'1', the dispatcher does not schedule the URL for Crawling.
If Status is'0', the dispatcher schedules the URL for Crawling.

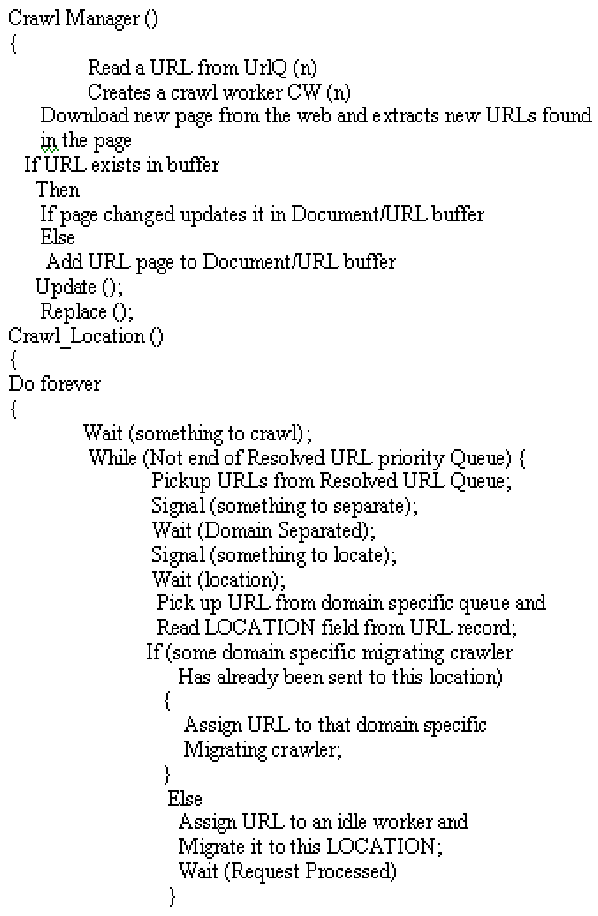
- **P1 and Pg:** are the boundary conditions i.e. upper and lower threshold values of Pc respectively.
- **The document pointer:** field contains the pointer to the original document.
- **FingerPrintKey:** fields store the finger print key value of the crawled page.

The proposed algorithms for the various modules of the Page Relevant Incremental Crawler are described below:

1. Crawler Module:
The crawling subsystem consists of following functional components:

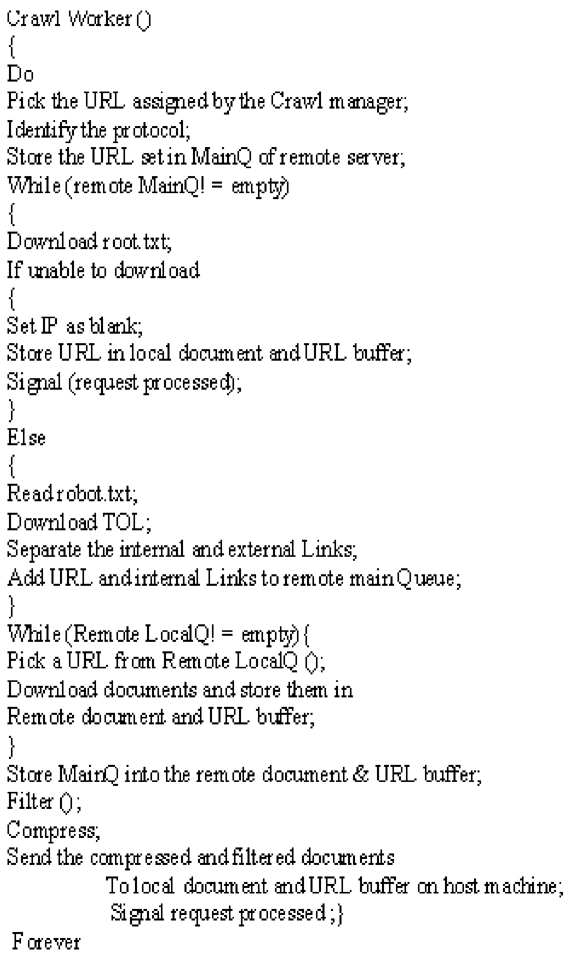
Crawl Manager: It creates multiple worker threads named as Crawl Workers.

Sets of resolved URLs from Resolved URL Queue are taken and each worker is given to domain separator who then separate the URLs based on their domain of nearest web server and each worker is given a domain specific location aware set of URLs. It sends a signal something to update to refresh time calculator module.



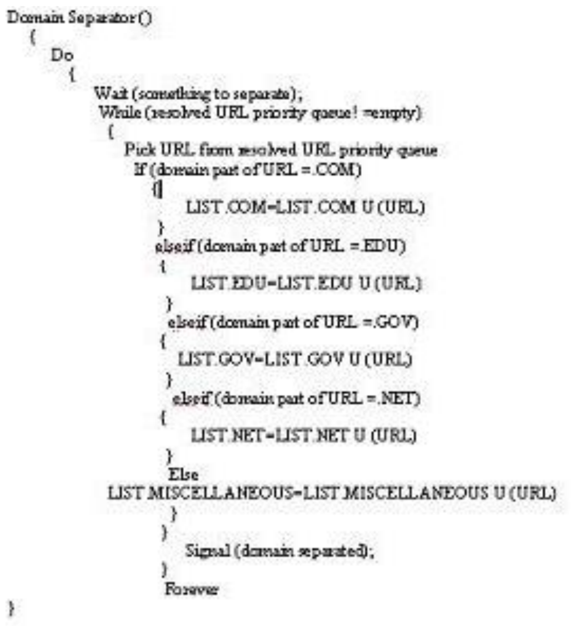
```
Crawl Manager ()
{
          Read a URL from UrlQ (n)
          Creates a crawl worker CW (n)
    Download new page from the web and extracts new URLs found
    in the page
  If URL exists in buffer
    Then
    If page changed updates it in Document/URL buffer
    Else
     Add URL page to Document/URL buffer
    Update ();
    Replace ();
Crawl_Location ()
{
Do forever
{
          Wait (something to crawl);
          While (Not end of Resolved URL priority Queue) {
                  Pickup URLs from Resolved URL Queue;
                  Signal (something to separate);
                  Wait (Domain Separated);
                  Signal (something to locate);
                  Wait (location);
                   Pick up URL from domain specific queue and
                   Read LOCATION field from URL record;
                 If (some domain specific migrating crawler
                     Has already been sent to this location)
                   {
                      Assign URL to that domain specific
                      Migrating crawler;
                   }
                    Else
                     Assign URL to an idle worker and
                     Migrate it to this LOCATION;
                     Wait (Request Processed)
                    }
```

2. Crawl Worker Module:
They are managed by Crawl Manager which supplies each crawl worker a list of target web site and monitors them .After downloading the document from the server it sends the signals request processed to crawl manager.



```
Crawl Worker ()
{
Do
Pick the URL assigned by the Crawl manager;
Identify the protocol;
Store the URL set in MainQ of remote server;
While (remote MainQ! = empty)
{
Download root.txt;
If unable to download
{
Set IP as blank;
Store URL in local document and URL buffer;
Signal (request processed);
}
Else
{
Read robot.txt;
Download TOL;
Separate the internal and external Links;
Add URL and internal Links to remote main Queue;
}
While (Remote LocalQ! = empty) {
Pick a URL from Remote LocalQ ();
Download documents and store them in
Remote document and URL buffer;
}
Store MainQ into the remote document & URL buffer;
Filter ();
Compress;
Send the compressed and filtered documents
             To local document and URL buffer on host machine;
             Signal request processed ;}
 Forever
```

3. Domain Separator Module:
The domain separator module takes the URL from resolved priority queue and separates them in different sets based on their domains. The algorithm for this is as follows:



```
Domain Separator()
  {
     Do
       {
          Wait (something to separate);
           While (resolved URL priority queue! =empty)
            {
              Pick URL from resolved URL priority queue
               If (domain part of URL =.COM)
                 {
                     LIST.COM=LIST.COM U (URL)
                 }
                elseif (domain part of URL =.EDU)
                 {
                      LIST.EDU=LIST.EDU U (URL)
                  }
                  elseif (domain part of URL =.GOV)
                 {
                    LIST.GOV=LIST.GOV U (URL)
                  }
                   elseif (domain part of URL =.NET)
                 {
                      LIST.NET=LIST.NET U (URL)
                  }
                  Else
          LIST.MISCELLANEOUS=LIST.MISCELLANEOUS U (URL)
                   }
                 }
                    Signal (domain separated);
                 }
                 Forever
}
```

4. Save/Update Module:

Save/Update module updates the database in search /insert method by replacing the less important pages with the more important page Update module one by one fetches the URLs and their updated link information (with update revisit frequencies) from the database..

The Save/Update Module makes use of various functions as mentioned below:

- Initialize (): To initialize all the variables, Initialize () is used.
- Update (): To check whether the page has been refreshed or not.
- Replace (): If the page has been updated, then Replace() replaces the less important pages with the more important pages in the database.
- Estimate_Freq (), New_maxf: To estimate the frequency and based on that frequency, to find out the max. Frequency.

3.a. Initialize Function: The Algorithm for Initialize Function is as follows;

5. Dispatcher:

It waits for the fetch URL signal from Update module and upon receiving this signal; it fetches an URL from the URL Buffer so that crawler can download the corresponding web pages. The algorithm for Dispatcher is as follows:

```
Dispatcher ()
{
 Wait (fetch URL)
 While (not (empty URL Buffer))
{
Fetch URL from Buffer;
Forward the URL to the crawler to download all the
web pages related to that URL.
```

6. URL Buffer:

It stores all the updated URLs which are need to be recrawled by the crawler. The URL Buffer is feeding by the Update Module.

7. Rank Updater:

Rank Updater checks the repository after a particular interval of time. It calculates the difference between the no. of user accesses of before updation which is denoted by a1 and after updation, denoted by a2 of the URLs. And if the difference between a1 and a2 is equal to the constant value that is decided by the administrator then it will be add the threshold value to the no. of user accesses of after updation of those URLs. Then rank updater updates the relevancy of the web pages according to it. The algorithm for the Rank Updater is as follows.

```
Rank Updater ()
{
 While (URLs) // accept URLs from Repository
{
Find the no_of _accesses a1 // before updation
Find the no_of _accesses a2 // after updation
Diff = a1 - a2
If Diff > τ // τ equals to constant value
Then
 Add threshold value to the no_of_accesses of after
updation of the URL.
```

8. Search Engine:

It is used to the proposed architecture for calculating the no. of user accesses. The no_of_accesses is calculated as follows:

```
Init () /* initialize variables */
N = 0; /* total number of accesses */
After first search
N = N + 1;
```

9. Repository:

The Repository stores all the web pages and related URLs that crawled by the crawler.

4. CONCLUSION & FUTURE WORK

The proposed architecture helps in maintaining the freshness of the repository and provides relevant web pages to the users. The calculation of the no. of user accesses helps in finding the importance of the web pages by efficiently managing the status checker so that the relevant pages are provided to the users. Moreover, the architecture is suitable for the applications where relevant search have been required.

Replace (): If the page has been updated, then Replace() replaces the less import

As the future work, the limitation of the current work that the crawler crawl the web pages even after the administrator stops the crawling process is taken into account. Future work includes applying a technique to overcome this problem.

5. REFERENCES

[1] Sakshi Goel, Anjana, Akhil Kaushik, Kirtika Goel, “A Novel Approach for Page Rank in Incremental Crawler”, IJCST Vol. 3, Issue 1, Jan. - March 2012.

[2] Niraj Singhal, Ashutosh Dixit, Dr. A. K. Sharma, “Design of a Priority Based Frequency Regulated Incremental Crawler”, 2010 International Journal of Computer Applications (0975 – 8887) Volume 1 – No. 1.

[3] Arvind Kumar, Km. Pooja, “An effective method for ranking of changed web pages in incremental crawler”, International Journal of Computer Applications (0975 – 8887) Volume 8– No.7, October 2010.

[4] Rosy Madaan, Ashutosh Dixit, A.K. Sharma, Komal Kumar Bhatia, “A Framework for Incremental Hidden Web Crawler”, International Journal of Computer Science and Engineering (IJCSNE), Vol. 02, No. 03, 2010.

[5] Ravita Chahar, Komal Hooda, Annu Dhankhar, “Management of Volatile Information In Incremental Web Crawler”, IJCSI International Journal of Computer Science Issues, Vol. 4, No. 1, 2009(ISSN (Online): 1694-0784, ISSN (Print): 1694-0814).

[6] 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.

[7] M.P.S.Bhatia, Divya Gupta, “Discussion on Web Crawlers of Search Engine”. Proceedings of 2nd National Conference on Challenges & Opportunities in Information Technology (COIT-2008) RIMT-IET, Mandi Gobindgarh. March 29, 2008.

[8] Cho, J. and Roy, “Impact of search engines on page popularity”. In Proc.13th International World Wide Web Conference, 2004.

[9] Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, S. Raghavan, “Searching the Web”, ACM Transactions on Internet Technology, Vol. 1, Num. 1, August 2001, pp.2-43.

[10] Mark Najork, Allan Heydon, “High- Performance Web Crawling”, September 2001.

[11] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In Proceedings of the 3rd ACM Workhop on Web Information and Data Management, pages 9–15, November 2001. Atlanta, USA.

[12] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan, “Searching the Web”, ACM Transactions on Internet Technology (TOIT), 1(1):2–43, August 2001.

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

[14] 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.

[15] 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.

[16] Henzinger M. R. Link analysis in web information retrieval. IEEE Data Engineering Bulletin, 23(3):3-8, September 2000.

[17] Jenny Edwards, Kevin McCurley, John Tomlin, “An Adaptive Model for Optimizing Performance of an Incremental Web Crawler”.

[18] Brin, Sergey and Page Lawrence, “The anatomy of a large-scale hypertextual Web search engine”. Computer Networks and ISDN Systems, April 1998.

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