Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 6, June 2014, pg.247 – 255

RESEARCH ARTICLE

An Elegant Fusion of Concurrent Crawling and Page Rank Technique for Spidering Websites

Smita Marwadi
M. Tech. scholar, Department of Computer Science and Engineering,
Rungta College of Engineering and Technology, BHILAI, (CG) INDIA
simi.marwadi@gmail.com

Mr. Neelabh Sao
Associate Professor, Department of Computer Science and Engineering,
Rungta College of Engineering and Technology, BHILAI, (CG) INDIA
neelabhsao@gmail.com

Abstract— The World Wide Web is expanding day by day. With the great growth of the Web, it has become a massive challenge for the all-purpose single process crawlers (A crawler is a program that downloads and stores Web pages, often for a Web search engine) to locate the resources that are precise and relevant in an appropriate amount of time, so more enhanced and convincing algorithms are in stipulate. Thus it becomes vital to improve the crawling procedure, in order to finish downloading pages in a sensible amount of time. Web crawler which employs multi-processing to allow multiple crawler processes to run concurrently. We have proposed a resourceful concurrent crawler that is fusion of page rank and concurrent multi-process crawler, offering a means to efficiently crawl the Web and presenting a scalable solution that allows crawl speeds to be tuned as needed.

Keywords- Web Crawler, Web Spider, concurrent crawler

I. INTRODUCTION

World Wide Web (WWW or Web) is a system of interlinked hypertext documents stored on servers all over the world [3], accessible through a number of protocols built on top of the internet architecture. In order to harvest this enormous data repository, search engines download parts of the existing web and offer Internet users access to this database through keyword search. One of the main components of search engines is web crawler. Web crawler is a web service that assists users in their web navigation by automating the task of link traversal, creating a searchable index of the web, and fulfilling searchers' queries from the index. That is, a web crawler automatically discovers and collects resources in an orderly fashion from the internet according to the user requirements. A crawler is an automated script, which independently browses the World Wide Web. It starts with a seed URL and then follows the links on each page in a Breadth First or a Depth First method [1]. But as the size of web is exponentially increasing, a more optimal scheme where multiple processes are running concurrent, downloading web pages independently by browsing the web [2].In particular, we believe the following issues make the study of a concurrent crawler challenging and interesting [4]:

1. A crawler must use as many resources as possible when crawling a specific site/server to allow the crawler to quickly and effectively deal with any content crawled and move onto other URLs or tasks. However at the same time, the crawler must not abuse a server hosting the crawled information by overloading it.
2. Crawlers similar to 'good pages'. Good pages can be defined as pages that are rich in information and hyperlinks. However, naturally a crawler isn't going to know whether or not a page is 'good' before requesting the resource from the server and analyzing it. This relates to point (1) mentioned above – adding a bottleneck to when other URLS located within the same domain can be crawled and wasting resources crawling effectively 'useless' pages to the crawler.
3. Copies of resources / pages must be fresh and up-to-date. However, determining how often to re-visit pages and crawl them is another issue. If pages are re-visited too often and the page hasn't changed resources are wasted. If pages aren't re-visited enough pages that exist within the crawler are outdated. There is also the issue that a crawler must have the ability to discover new pages also. Thus re-visiting pages and discovering new pages require being unbiased effectively.

In this paper we have proposed a new model and architecture for a Web crawler that tightly integrates the crawler with the rest of the search engine, providing access to the data and links of the documents that can be used to guide the crawling process effectively. To achieve such goal-directed crawling, we have designed a fusion of concurrent crawling and page ranking. Page rank algorithm determines the importance of the web pages by counting citations or backlinks to a given page.

The remainder of this paper is organized as follows: Section 2 provides a brief review of the Background. In Section 3, we drew the attention towards problems in existing systems. In Section 4, we introduce our proposed architecture. An illustration of the algorithm is presented in section 5. Section 6 demonstrates the experimental results with some screen shots. Finally, we concluded our work.

II. BACKGROUND

A. WEB CRAWLERS:

Existing Web Crawlers: - When using the term "Web Crawler" most people would more than likely think of the most popular site on the web Google. However, there are also several other large-scale crawlers such as: Microsoft Bing, Internet Archive, Yahoo. There are also several open-source implementations for large-scale crawling such as: Apache Nutch1, ABot2 and Heritrix3.

Majority of the larger-- - scale web crawlers are generally used as the background processing for search engines. Indexing and ranking pages based on their content quality and returning the correct information and results from search queries is a complicated and resource intensive task – hence why they're probably the most appreciated in terms of web crawling.

However not all crawlers are design to cover the entire web in a "general" fashion. Crawlers such as the Heritrix are designed to crawl the entire web and mirror exactly what it discovers making it a crawler that is designed to download not only web-pages but other media types such as images and zip archives. Although there are open-source implementations regarding web crawling, majority of the large-scale web crawler solutions are "business secrets" making the competition to build an exceptional crawler all the more difficult. Since the web is growing an increasingly fast rate, there are billions of web pages to process. However, the number of URL's pointing to these billions of web pages greatly exceeds the number of web pages that exist, which is why it is important to design a structurally efficient web crawler.

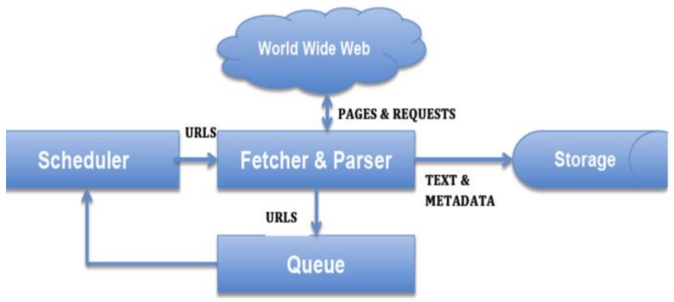


Figure 1. [4]A general overview of a web crawler's architecture.

As a general overview, [4] a web crawler looks relatively simple. High-level components include:

- **Fetcher & Parser** – Downloads & parses downloaded content.
- **Storage** – Form of storage method for storing URLs and any specifically crawled content.
- **Queue** – a queue of URLs ready to be crawled by the crawler.
- **Scheduler** – a method of scheduling URLS (can either be based on content, timeout periods or other factors)

B. PAGE RANK

The original formula for PageRank due to [9], is a summation formula which calculates the popularity of the pages by adding up the PageRank of all the pages pointing to this web page. Page rank algorithm determines the importance of the web pages by counting citations or backlinks to a given page [9]. The page rank of a given page is calculated as

$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))$

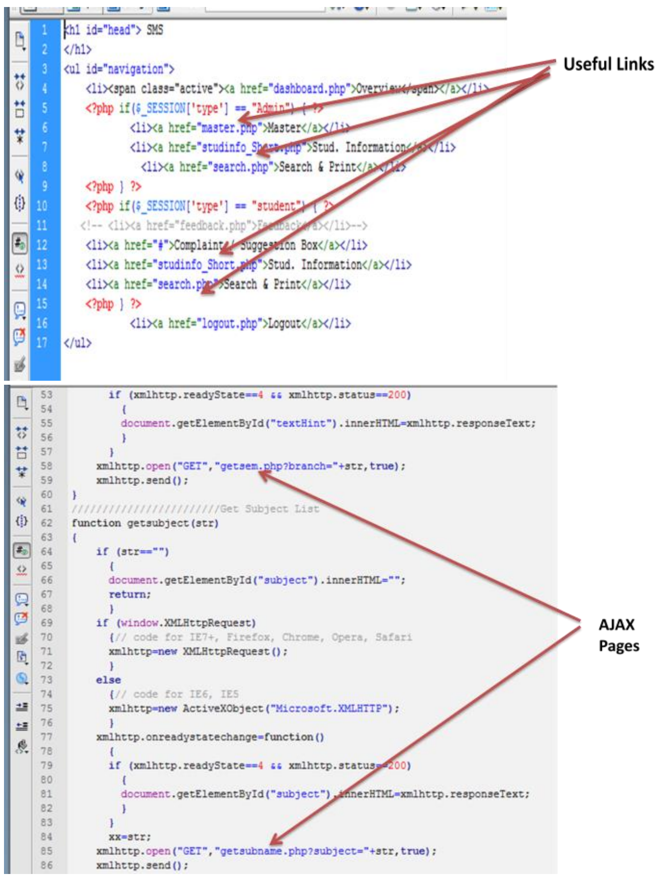
PR(A):- Page Rank of a Website,
d :-damping factor
T1,….Tn :- links

III. PROBLEM IDENTIFICATION

Most existing crawlers use local search algorithms in Web searching. Using such algorithms, a crawler will miss a relevant page if there do not exist a chain of hyperlinks that connects one of the starting pages to that relevant page.

Furthermore, unless the hyperlinks on the chain all point to relevant pages, the crawler will give up searching in this direction before it reaches the final target. Because of this limitation, crawlers using local search algorithms can only find relevant pages within a limited sub-graph of the Web that surrounds the starting URLs and any relevant pages outside this sub-graph will be ignored. Problems in existing techniques are as follows:

- Circular Reference.
- Due to swift growth of the World Wide Web poses unprecedented scaling challenges for the all-purpose single-process crawlers.
- At the same time of maximizing the coverage rate of web resources, General Crawlers also download a large amount of useless information.
- Most of General Crawlers only support keywords search, but not the attribute search.
- A large amount of web pages are written in JavaScript or Ajax. It's impossible to extract new URLs by tag matching because these link URLs are generated by JavaScript functions.



```
<h1 id="head"> SMS
</h1>
<ul id="navigation">
    <li><span class="active"><a href="dashboard.php">Overview</span></a></li>
    <?php if($_SESSION['type'] == "Admin") { ?>
            <li><a href="master.php">Master</a></li>
            <li><a href="studinfo_Short.php">Stud. Information</a></li>
              <li><a href="search.php">Search & Print</a></li>
    <?php } ?>
    <?php if($_SESSION['type'] == "student") { ?>
   <!-- <li><a href="feedback.php">Feedback</a></li>-->
    <li><a href="#">Complaint / Suggestion Box</a></li>
    <li><a href="studinfo_Short.php">Stud. Information</a></li>
    <li><a href="search.php">Search & Print</a></li>
    <?php } ?>
            <li><a href="logout.php">Logout</a></li>
</ul>
```

```
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
          {
          document.getElementById("textHint").innerHTML=xmlhttp.responseText;
          }
        }
    xmlhttp.open("GET","getsem.php?branch="+str,true);
    xmlhttp.send();
}
////////////////////////Get Subject List
function getsubject(str)
{
    if (str=="")
      {
      document.getElementById("subject").innerHTML="";
      return;
      }
    if (window.XMLHttpRequest)
      {// code for IE7+, Firefox, Chrome, Opera, Safari
      xmlhttp=new XMLHttpRequest();
      }
    else
      {// code for IE6, IE5
      xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
    xmlhttp.onreadystatechange=function()
      {
      if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
        document.getElementById("subject").innerHTML=xmlhttp.responseText;
        }
      }
      xx=str;
    xmlhttp.open("GET","getsubname.php?subject="+str,true);
    xmlhttp.send();
```

Figure 2. Useful and Un-useful links

IV. PROPOSED ARCHITECTURE

In this proposed architecture, the Web is partitioned among a number of software programs, called Web crawlers. Crawler is responsible for downloading a subset of pages on the Web. The crawlers locate the pages by following the hyperlinks among the pages. Instead of using normal crawler we are using Multithreaded Crawler. As the size of the Web grows, it becomes imperative to parallelize a crawling process, in order to finish downloading pages in a reasonable amount of time. Web crawler which employs multi-processing to allow multiple crawler processes to run in parallel. While it is possible to run crawlers on multiple threads, such a solution lacks the scalability desired for the system, since threads can only be spawned on a single machine, limiting the total number of concurrent crawlers possible. An indexer is responsible for converting the documents into a queryable form, which is often an inverted index. We have done indexing using topic sensitive page raking algorithm. The users submit queries to the retrieval system through a user interface. The queries are then submitted to index servers. Index servers access their local disks, determine the set of documents matching the query, and send these answer sets back to the interface. Finally, the user is returned a set of best-matching documents.

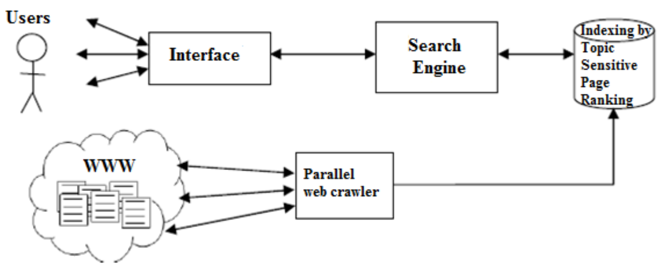


Figure 3. Architecture of Proposed System

A sequential crawling loop spends a large amount of time in which either the CPU is idle (during network/disk access) or the network interface is idle (during CPU operations). Multi-threading, where each thread follows a crawling loop, can provide reasonable speed-up and efficient use of available bandwidth. Figure 4 shows our multi-threaded version of the basic crawler. Note that each thread starts by locking the URL list to pick the next URL to crawl. After picking a URL it unlocks the URL list allowing other threads to access it. The URL list is again locked when new URLs are added to it. The locking steps are necessary in order to synchronize the use of the URL list that is now shared among many crawling loops (threads). The model of multi-threaded crawler in Figure 4 follows a standard parallel computing model.

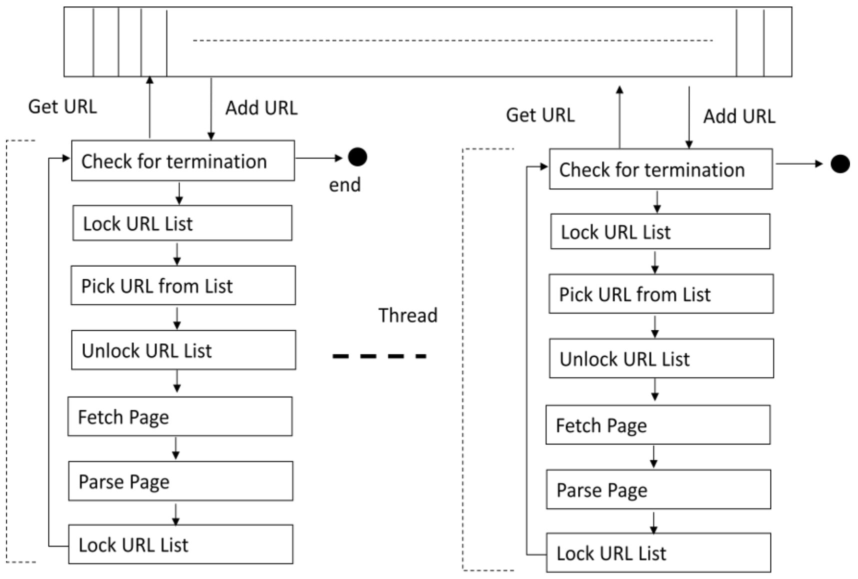


Figure 4. Model for Concurrent Crawler

Crawling can be viewed as a graph search problem as shown in figure 5. The Web is seen as a large graph with pages at its nodes and hyperlinks as its edges. A crawler starts at a few of the nodes (seeds) and then follows the edges to reach other nodes. The process of fetching a page and extracting the links within it is analogous to expanding a node in graph search A topical crawler tries to follow edges that are expected to lead to portions of the graph that are relevant to a topic.

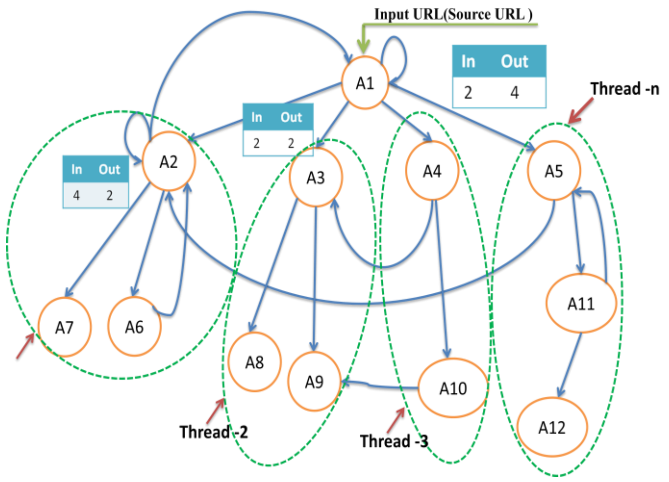


Figure 5. Representation of Crawling Process as a Graph Search

V. Crawling Algorithm

A crawler for a large search engine has to address two issues. 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. To retrieve all webpage contents, the HREF links from every page will result in retrieval of the entire web's content. The basic steps of crawling process are as follows:

1. Remove a URL from the unvisited URL list.
2. Determine the IP Address of its host name.
3. Download the corresponding document.
4. Extract any links contained in it.
5. If the URL is new, add it to the list of unvisited URLs.
6. Process the downloaded document.
7. Back to step 1

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. 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.

Web Crawler Algorithm

```
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
```

Algorithm for Avoid Circular Reference

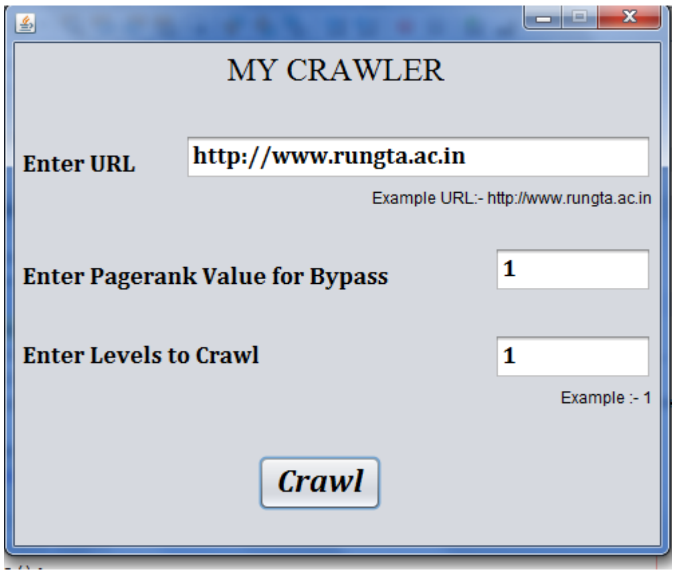
```
1.  Initialize two set s1,s2

            //s1- Set of Crawled URL
            //s2 – Generated URL

2.  s1 ← {Ø}
3.  s2 ← {u1}//u1- Input URL
4.  s2 ← s1
5.  Repeat step 6-9
6.  Crawl URL's of s2 and assign to s2
7.  if s2= {Ø} then stop
8.  else s3=s2-s1
9.  s1=s1Ụs3
10. end
```

VI. Experimental results with screenshots

This section demonstrates a simple walk-through of home page approach which contains the major links like Home, crawler graph and output file. Web Crawler system exploits the graph structure of the Web to move from page to page.


MY CRAWLER

Enter URL: http://www.rungta.ac.in

Example URL:- http://www.rungta.ac.in

Enter Pagerank Value for Bypass: 1

Enter Levels to Crawl: 1

Example :- 1

Figure 5: Home Page of the System

The Fig. 5 displays home page which contains tabs like Enter URL, Enter PageRank Value for bypass and enter levels to crawl. This system provides the facility to the users that they can add levels and PageRank values according to their need.

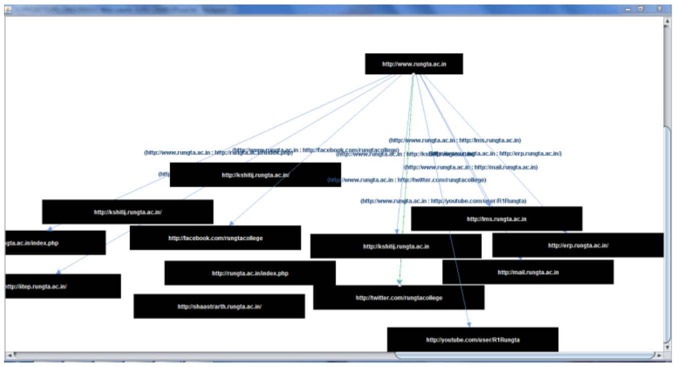
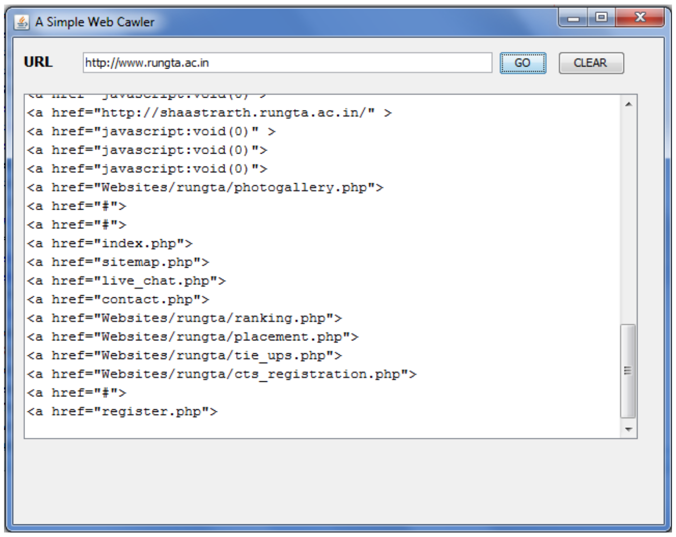

http://www.rungta.ac.in
(http://www.rungta.ac.in : http://lms.rungta.ac.in)
(http://www.rungta.ac.in : http://facebook.com/rungtacollege)
(http://www.rungta.ac.in : http://erp.rungta.ac.in/)
(http://www.rungta.ac.in : http://mail.rungta.ac.in)
(http://www.rungta.ac.in : http://twitter.com/rungtacollege)
(http://www.rungta.ac.in : http://youtube.com/user/R1Rungta)
http://kshitij.rungta.ac.in/
http://kshitij.rungta.ac.in/
http://lms.rungta.ac.in
http://facebook.com/rungtacollege
http://kshitij.rungta.ac.in
http://erp.rungta.ac.in/
http://rungta.ac.in/index.php
http://mail.rungta.ac.in
http://itep.rungta.ac.in/
http://twitter.com/rungtacollege
http://shaastrarth.rungta.ac.in/
http://youtube.com/user/R1Rungta
Figure 6: Output screen for graph structure of the crawler

Fig 6 shows graph output after applying the algorithm. It shows all the link structure associated with the web site.



```
<a href="http://shaastrarth.rungta.ac.in/" >
<a href="javascript:void(0)" >
<a href="javascript:void(0)">
<a href="javascript:void(0)">
<a href="Websites/rungta/photogallery.php">
<a href="#">
<a href="#">
<a href="index.php">
<a href="sitemap.php">
<a href="live_chat.php">
<a href="contact.php">
<a href="Websites/rungta/ranking.php">
<a href="Websites/rungta/placement.php">
<a href="Websites/rungta/tie_ups.php">
<a href="Websites/rungta/cts_registration.php">
<a href="#">
<a href="register.php">
```

Figure 7: Output file of the link structure

Fig 7 shows the file that contains the crawler output in the form of text data. It contains all the links of the crawled site.

VII. CONCLUSION

World Wide Web is a extremely powerful resource. It contains a vast amount of related and unrelated information. Hence, there is a great requirement to have algorithms that could list relevant web pages accurately and efficiently on the top of few pages. Despite the vast amount of both theoretical and practical research on information retrieval, the search problem is still far from being solved. Crawlers are being used more and more often to collect Web data for search engine, caches, and data mining. In this work, we aimed to put just another small brick into the wall of research on information retrieval. In particular, we proposed models and algorithms for efficient meta crawling.. We have combined the concurrent crawler with the Page Rank with a view to resolve the existing problems. This algorithm will improve the order of web pages in the result list so that user may get the relevant pages easily.The theoretical results indicate that the proposed models are quite successful in obtaining an effective utilization of system resources during the query processing. By taking into consideration we have mainly focused on designing of web crawling system for efficiently crawling of web sites based on the user preference regarding page rank and level of crawling. Experiments conducted on various web sites show the viability of our approach. We currently conduct studies to evaluate the performance of the proposed Web crawling model in practice. In future we endeavor to design an architecture which can continuously update and refresh the web information and update the repository periodically.

REFERENCES

[1] David Eichmann, “The RBSE Spider – Balancing effective search against web load”, Repository Based Software Engineering Program , Research Institute for Computing and Information Systems, University of Houston – Clear Lake.
[2] Junghoo Cho & Hector Garcia-Molina, “Parallel Crawlers”. Proceedings of the 11th international conference on World Wide Web WWW '02, Honolulu, Hawaii, USA. ACM Press. Page(s): 124 – 135
[3] Md. Abu Kausar and V. S. Dhaka "An Effective Parallel Web Crawler based on Mobile Agent and Incremental Crawling" Journal of Industrial and Intelligent Information Vol. 1, No. 1, March 2013.
[4] Michael GrahamInterim Report Concurrent Thread based Web Crawler.
[5] Diligenti, M., Coetzee, F., Lawrence, S., Giles, C.L. &Gori, M. 2000. Focused crawling using context graphs. Proc. of the 26th International Conference on Very Large Databases (VLDB), Cairo, Egypt, 2000, pp. 527-534.
[6] Chakrabarti, S. 2003. Mining the Web: Discovering Knowledge from Hypertext Data. San Francisco, CA, USA: Morgan Kaufmann Publishers.
[7] Amento, B., Loren, T. & Hill, W., 2000. Does “authority” mean Quality? Predicting expert quality ratings of Web documents. In Proceedings of SIGIR 2000, Athens, Greece.
[8] Taher H. Haveliwala. Effciient computation of PageRank. Stanford University Technical Report, 1999.
[9] Sergey Brin and Lawrence Page “Anatomy of a Large scale Hypertextual Web Search Engine” Proc. WWW conference 2004
[10] Spark Jones, K., Walker, S. & Robertson, S. E., 2000. A probabilistic model of information retrieval: Development and comparative experiments (Parts 1 & 2). Information Processing and Management, Vol. 36, Issue 6, pp. 779-808, 809-840.
[11] Haveliwala, T.H. 2002. Topic-sensitive PageRank. Proc. of the 11th International WWW Conference, Honolulu, Hawaii, USA (May 6-11, 2002)