PARCAHYD: An Architecture of a Parallel Crawler based on Augmented Hypertext Documents

A. K. Sharma
Department of Comp. Engg.
YMCA University of Science and Technology,
Faridabad , India
ashokkale2@rediffmail.com

J.P. Gupta
Jaypee Institute of Information Technology, Noida, India
prof_jpgupta@gmail.com

D. P. Agarwal
Indian Institute of Information Technology & Management, Gwalior, India
prof_dpa@hotmail.com

Abstract

Search engines use web crawlers to collect documents for storage, indexing and analysis of information. Due to the phenomenal growth of web, it becomes vital to create high performance crawling systems. Augmentations to hypertext documents were proposed [6] so that the documents become suitable for parallel crawlers. PARCAHYD is an on going project aimed at designing of a Parallel Crawler based on Augmented Hypertext Documents. In this paper, the architecture of this parallel crawler is presented.

Keywords : *Search Engines, Web Crawlers, Parallel Crawlers, PARCAHYD*

1. Introduction

The World Wide Web (WWW)[1,18] is internet client server architecture. It is a powerful mechanism based on full autonomy to the server for serving information available on the internet. The information is organized in the form of a large, distributed, and non-linear text system known as Hypertext Document [2] system. This system defines portions of a document as being hypertext- pieces of text or images which are linked to other documents via anchor references. HTTP and HTML [3] provide a standard way of retrieving and presenting the hyperlinked documents. Client applications i.e. Internet browsers, use search engines to search the Internet servers for required pages of information. The pages supplied by the server are processed at the client side.

Due to the extremely large nature of the pages present on web, search engines depend upon crawlers [4] for the collection of pages. A crawler follows hyperlinks present in the documents to download and store the pages in the database of the search engine. The search engine indexes the pages for later on manipulations of the user queries.

The web has more than 350 million pages and is growing in the tune of one million pages per day. Such enormous growth and flux necessitates the creation of highly efficient crawling system [5,12,13]. Research is being carried out in the following areas:

- to develop strategies to crawl only the relevant pages
- to design architectures for parallel crawlers
- restructuring of hypertext documents

We proposed augmentation to the hypertext documents [6] so that they become suitable for downloading by parallel crawlers. The augmentations do not affect the current structure of hypertext system. This paper discusses hypertext documents, the proposed augmentations, and the design of an architecture of a parallel crawler based on the augmented hypertext documents (PARCAHYD). The implementation of this crawler in Java is in progress.

2. Related work

A program that indexes, automatically navigates the web, and downloads web-pages is called a web crawler [4,14]. It identifies a document by its Uniform Resource Locator (URL). From the URL, the crawler can search and downloads the document as per the algorithm given below:

```
Crawler ()
   Begin
      While (URL set is not empty)
         Begin
           Take a URL from the set of seed URLs;
           Determine the IP address for the host name;
           Download the Robot.txt file which carries downloading permissions and also specifies the files to be
           excluded by the crawler;
           Determine the 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;
   End.
```

The Google search engine employs a crawler consisting of five functional components [3]. The components run in different processes listed below:

- ***URL server process***: takes URLS from a disk file and distributes them to multiple crawler processes.
- ***Crawler Processes***: fetches data from web servers in parallel. The downloaded documents are sent to Store server process.

- ***Store server process***: compresses the documents and stores them on a disk.
- ***Indexer process***: takes the pages from the disk and extracts links from the HTML pages and saves them in a different file called a link file.
- ***URL resolver process***: reads the links from the file, resolves their IP addresses, and saves the absolute URLs to disk file.

Internet Archive [1,3] uses multiple crawler processes to crawl the web. Each crawler process is single threaded which takes a list of seed URLs and fetches pages in parallel. The links are extracted from the downloaded documents and placed into different data structures depending upon the nature of their links i.e. internal and external links.

Mercator [7, 15] is a scalable and extensible web crawler. It uses the following functional components:

- ***URL frontier*** : for storing the URLs.
- ***DNS resolver***: for resolving host names into IP addresses.
- ***Downloader component***: downloads the documents using HTTP protocol.
- ***Link extractor***: for extracting links from HTML documents.
- ***Content Seen Process***: to check whether a URL has been encountered before.

Jungoo Cho [3] has suggested a general architecture of a parallel crawler. It consists of multiple crawling processes. Each process is called as C-Proc. Each C-proc performs the tasks of which a single processes crawler conducts. It downloads pages from the web, stores the pages locally, extracts URLs from the downloaded pages and follows the links

A critical look at the available literature indicates that in the current scenario, the links become available to the crawler only after a document has been downloaded. Hence this is a bottleneck at the document level from parallel crawling point of view. None of the researchers have looked into this aspect of the document as one of the factors towards delay in overlapped crawling of related documents.

2.1 The Augmented Hypertext documents

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 was proposed [6] 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 (say .TOL). This one time extraction of TOL can be done at the time of creation of the document. The algorithm for the extraction of TOL from a hypertext documents was also provided [6].

3. The Architecture of PARCAHYD, the Parallel Crawler

The art of parallelism is to divide a task into subtasks which may execute, at least partially independently, on multiple processing nodes [8]. This decomposition can be based on the process, the data, or some combination.

At the first stage, we have divided the document retrieval system into two parts: the crawling system and the hypertext (augmented) documents system. The augmented hypertext documents provide a separate TOL for each document to be downloaded by the crawling process. Once the TOL of a document becomes available to the crawler, the linked documents, housed on external sites, can be downloaded in parallel by the other instances of the crawler. Moreover, the overlapped downloading of the main documents along with its linked documents on the same site also becomes possible.

At the second stage, the crawling system has been divided into two parts: Mapping Process and Crawling Process. The Mapping process resolves IP addresses for a URL and Crawling Process downloads and processes documents.

3.1 The Mapping Process

The mapping process, shown in Fig.1, consists of the following functional components:

- ***URL-IP Queue****:* It consists of a queue of unique seed URL-IP pairs. The IP part may or may not be blank. It acts as an input to the Mapping Manager.
- ***Database****:* It contains a database of downloaded documents and their URL-IP pairs. The structure of its table consists of the following fields:
    - URL
    - IP-Address
    - Document –ID
    - Length
    - Document
- ***Resolved URL-Queue:*** It stores URLs which have been resolved for their IP addresses and acts as an input to the Crawl Manager.
- ***URL Dispatcher:*** This component reads the database of URLs and fills the URL-IP Queue. It may also get initiated by the user who provides a seed URL in the beginning. It sends a signal: *Something to Map* to the Mapping manager. Its algorithm is given below:

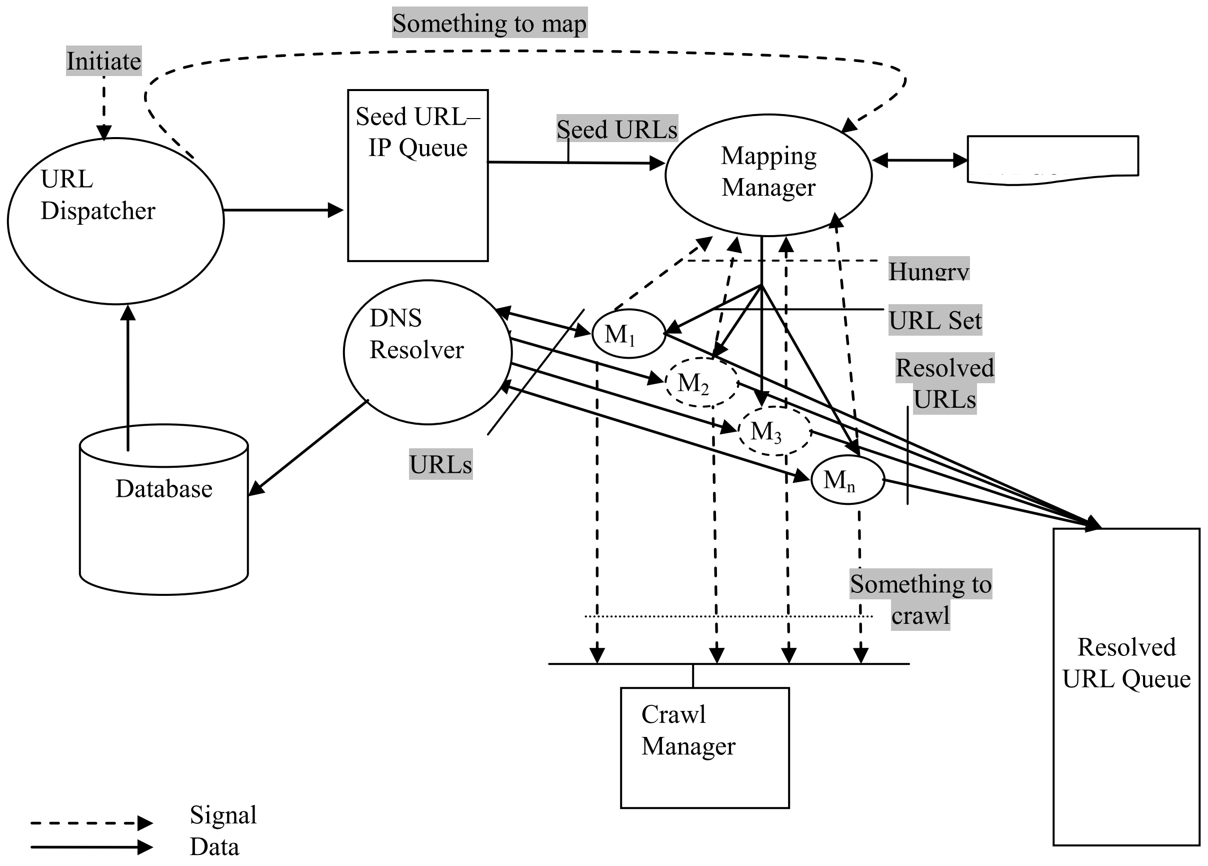

Fig. 1. The Mapping Manager

```
URL_Dispatcher ( )
    Begin
        Do Forever
            Begin
                While (URL-IP Queue not Full)
                    Begin
                        Read URL-IP pair from Database;
                        Store it into URL-IP Queue;
                    End;
                    Signal (Something to Map);
            End;
    End;
```

- ***DNS Resolver**:* Generally the documents are known by the domain names of their servers. The name of the server must be translated into an IP address before the crawler can communicate with the server[15]. The internet offers a service that translates domain names to corresponding IP addresses and the software that does this job is called the Domain Name System (DNS). The DNS resolver uses this service to resolve the DNS address for a URL and returns it back to the calling URL Mapper. It then updates the database for the resolved IP address of URL.

- *MapConf.Txt:* It is a mapper configuration file which is used by the Mapping Manager to load the initializing data. The contents of a sample file are tabulated in Table 1.

Table 1 : Contents of MapConf.Txt

Name	**Value**	**Description**
DbUrl	jdbc:oracle:thin:@myhost:1521:orcl	The database URL
DbName	crawldb	The database Name
DbPassword	crawldb	The database Password
DnsResloverClass	dnsResloverClass	The DNS RESOLVER CLASS name. This component is a pluggable component. Any third party component can be used. The name of class will be registered here and using this name the URL Mapper would instantiate this component.
MaxInstances	5	The maximum no. of instances to be created for URL Mapper component
LocalInstance	No	The instances to be created on same (local) or different machine
ListIP	135.100.2.98, 135.100.2.29, 135.100.2.28	If different then IP detail of those machines
ArgumentUrl	10	The maximum no. of URLs in a set to be given as arguments to an instance

- ***Mapping Manager:*** This component reads MapConf.txt. After receiving the signal *Something to Map*, it 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. Its algorithm is given below:

```
Mapping_manager ( )
Begin
        Read MapConf.Txt;
        Create instances of URL Mapper $M_1$ to $M_n$;
        Do Forever
                Begin
                    Wait (Something to map);
                    While (URL-IP Queue not empty)
                        Begin
                                Wait (hungry);
                                Pickup seed URLs from URL-IP Queue;
                    Assemble a set of URL-IPs;
                    Assign the set to an idle URL_Mapper;
                End;
            End;
    End.
```

- ***URL Mapper:*** 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. Its algorithm is given below:

```
URL-Mapper ( )
    Begin
        Do forever
            Begin
              While (URL-IP set is not empty)
                Begin
                  Take a URL-IP pair from the set;
                  If the IP is blank
                    Then
                      Begin
                        Call DNS resolver to resolve URL for IP;
                        Wait for the resolved URL;
                      End;
                  Store the Resolved URL in the Resolved URL Queue;
                  Signal (something to crawl);
                End;
                  signal (Hungry);
            End;
    End.
```

3.2 The Crawling Process

The crawling process as shown in Fig. 2 consists of the following functional components:

- ***WorkConf.txt:*** It is a worker configuration file which is used by the Crawl Manager to load the initializing data. The contents of a sample file are tabulated in Table 2.

- ***Crawl Manager:*** This component waits for the signal *something to crawl*. It reads the WorkConf.txt and as per the specifications stored in the file, it creates multiple worker threads named as Crawl Workers. Sets of resolved URLs from Resolved URL Queue are taken and each worker is given a set. Its algorithm is given below:

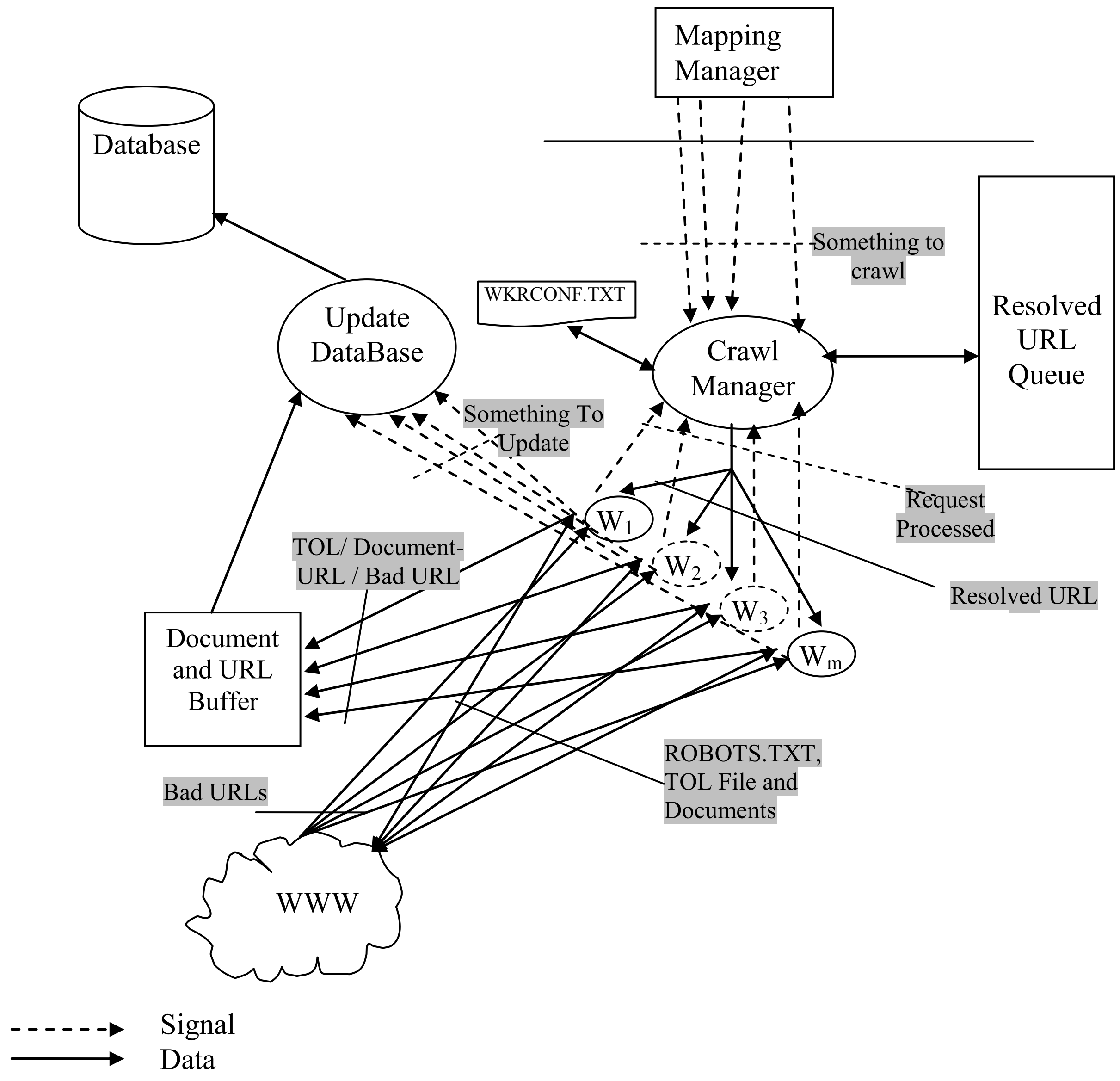

Fig. 2 The Crawl Manager

```
Crawl_Manager ( )
        Begin
                Read WorkConf.txt file;
                Create multiple instances of Crawl Workers: $W_1$ to $W_m$;
          Do forever
              Begin
                    Wait (something to crawl);
                    While (Not end of Resolved URL Queue)
                         Begin
                                 Wait (request processed);
                Pickup URLs from Resolved URL Queue;
                Assemble and assign a set of URLs to an idle crawl worker;
            End;
        End;
    End.
```

Table 2 ; Contents of WorkConf.txt

Name	Value	Description
DbUrl	jdbc:oracle:thin:@myhost:1521:orcl	The database URL
DbName	crawldb	The database Name
DbPassword	crawldb	The database Password
DownloaderClass	downLoaderClass	The DOWNLOADER CLASS name. This component is a pluggable component. Any third party component can be used. The name of class will be registered here and using this name the Worker threads would instantiate this component
MaxInstances	10	The maximum no. of instances to be created for URL Mapper component
LocalInstance	YES	The instances to be created on same (local) or different machine
ListIP	localhost	If different then IP detail of those machines
ArgumentUrl	5	The maximum no. of URLs to be given as arguments to an instance

- ***Crawl Worker:*** It maintains two queues: MainQ and LocalQ. The set of URLs received from crawl Manager is stored in MainQ(see Fig. 3). It down loads the documents as per the algorithm given below:

```
Crawl_Worker ( )
    Begin
        Store the URL set in MainQ;
        While ( MainQ is not empty)
        Begin
                Pickup a URL;
                Identify its protocol;
                Download robot.txt;
                If unable to download
                    Begin
                        Set IP part as blank;
                        Store URL in Document and URL Buffer;
                        Signal (something to update);
                    End;
                Else
                    Begin
                        Read robot.txt;
                        Download TOL;
                        Segregate the internal and external Links;
                        Add the URL and internal Links to LocalQ;
                        Store the External Links in the
                        Document and URL buffer;
                        Signal ( something to update);
                    End;
                While (LocalQ is not empty)
                    Begin
                        Pickup a URLfrom LocalQ;
                        Download document;
                        Store the document and its URL in
                                Document and URL
Buffer;
                        Signal (something to Update);
                    End;
            Signal (request processed);
        End;
    End.
```

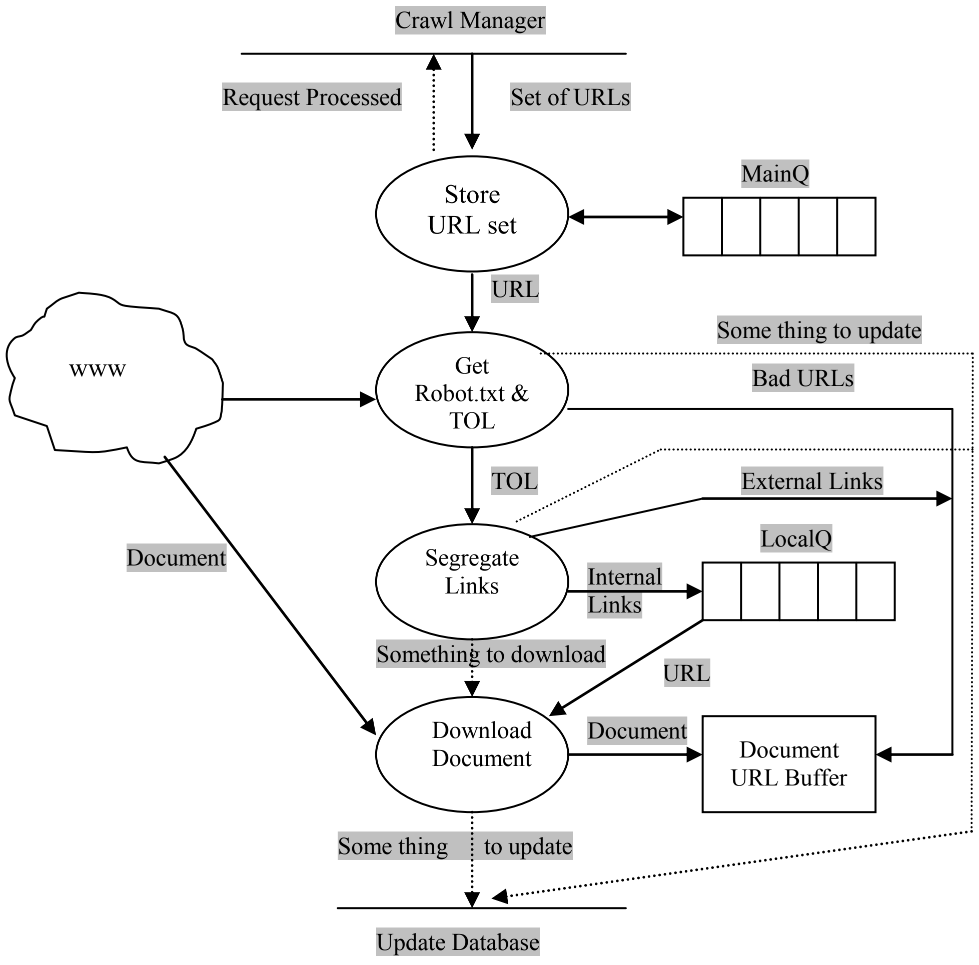

Fig. 3 Crawl Worker

- ***Document and URL Buffer:*** It is basically a buffer between the crawl workers and the Update Database Process It consists of following three Queues:
  - External Links queue (ExternLQ): This queue stores the external links.
  - Document-URL queue (DocURLQ): This queue stores the documents along with there URLs.
  - Bad-URL queue (BadURLQ): This queue stores all the resolved Bad-URLs.
- **Update Database:** This process waits for the signal *something to update* and on receiving the same, updates the database with the contents of the Document and URL Buffer. In order to use storage efficiently, each document is compressed using zlib [9] algorithm. The algorithm of Update Database Process is given below:

```
update database()
        Begin
        Set MaxSize to the maximum size of a batch;
        Do forever
          Begin
                Wait (something to update);
                        No-of-records = 0;
                        While (ExternLQ is not empty & No-of-records < MaxSize)
                                Begin
                                        Pickup an element from ExternLQ;
                                        Add to the batch of records to be updated;
                                        No-of-records= No-of-records+1;
                                End;
                        Update batch to database;
                        If (Updation is unsuccessful)
                                Then write batch to ExternLQ;

                        No-of-records = 0;
                        While (DocURLQ is not empty & No-of-records < MaxSize)
                                Begin
                                        Pickup an element from DocURLQ;
                                Compress the document;
                                        Add to the batch of records to be updated;
                                        No-of-records= No-of-records+1;
                                End;
                        Update batch to database;
                        If (Updation is unsuccessful)
                                Then write batch to DocURLQ;

                        No-of-records = 0;
                        While (BadURLQ is not empty & No-of-records < MaxSize)
                                Begin
                                        Pickup an element from BadURLQ;
                                        Add to the batch of records to be updated;
                                        No-of-records= No-of-records+1;
                                End;
                        Update batch to database;
                        If (Updation is unsuccessful)
                               Then write batch to BadURLQ;


                  End;

            End.
```

It may be noted here that each crawl worker independently downloads documents for the URL set received from Crawl Manager. Since all workers use different seed URLs, we hope that there will be minimum overlap of downloaded pages. Thus, the architecture requires no coordination overheads among the workers rendering it to be highly scalable system.

The PARCHAHYD has been used as a tool to test the following schemes / mechanisms for parallel crawling of documents from the web.

- A bottleneck towards parallel download of a document and its related documents was identified in the sense that in the current scenario, the links become available to the crawler only after the document has been downloaded. This bottleneck prohibits the document from parallel crawling of the related document on the same site or at remote site. None of the researchers have looked into this aspect of the document as one of the factors towards delay in overlapped crawling of related documents.

- A mechanism was devised to remove the bottleneck. The crawler is provided with the links to related documents in the form of a table of links (TOL) [6] even before the main document itself is downloaded. The TOL containing the meta-information is stored in the form of a separate file with the same name as of the document but with TOL as the extension. This eager strategy enables parallel crawling of the main document and its related documents. The experiments done at the Intranet of A. B. Indian Institute of Information Technology and Management, Gwalior, establishes the purpose that PARCAHYD takes 15.1% less time than an identical crawler without TOL. In fact, any other parallel crawler can also make use of this mechanism for eager crawling of related documents.

- Due to mirroring of the documents or different URLs pointing to the same document [18], a crawler may download several copies of the document. A novel scheme to reduce the duplication of the document has been devised [10] wherein a 64 bit document fingerprints (DF) for a static document is generated. The DF guaranteed that if two fingerprints were different then the corresponding two documents were definitely different. It has been proved that the probability of matching the signature of two different documents is $(1/256)^{16}$ which is very small. The scheme was tested on 16544 files and the results have been reported. The scheme drastically reduces the redundancy at the search engine side.

- The web is changing at very high rate. In dynamic documents, the contents contain volatile information only at particular places. Such documents put tremendous pressure on network traffic rendering the crawling process helplessly slow and inefficient. A novel scheme for the management of the volatile information was devised [11]. The <SPAN> Tag of HTML document were used to specify pieces of text as volatile information. Similarly, the user defined <Vol#> tags were chosen in the case of XML document for exactly the same purpose.

The tags and the information were stored separately in a file having same name but with **.**TVI extension. The table of variable information (TVI) file is definitely substantially smaller in size as compared to the whole document. The crawler needs to bring the **.**TVI file only to maintain fresh information at the search engine site. The .TVI file is used to update the main document either in the transaction processing manner or during the parsing of the document while displaying it on the screen. The scheme was tested at the intranet of YMCA Institute of Engg. Faridabad. Storing information in a data base is a computationally expensive exercise especially when a document contains only a small fraction of volatile information. Downloading of needless data has been substantially reduced by bringing only the .TVI file instead of the whole document itself. This scheme can also be used by any other crawler to reduce the network traffic and needless database operations as well. The documents, which use above scheme of storage were named as *augmented hypertext documents*.

4. Conclusions

Parallelization of crawling system is very vital from the point of view of downloading documents in a reasonable amount of time. The work done reported herein focuses on providing parallelization at three levels: the document, the mapper, and the crawl worker level. The bottleneck at the document level has been removed. The efficacy of DF (Document Fingerprint) algorithm and the efficiency of volatile information has been tested and verified. PARCAHYD is designed to be scalable parallel crawler. This paper has enumerated the major components of the crawler and their algorithmic detail has been provided. JAVA 2 has been chosen to make it platform independent. Besides the PARCHAHYD, it has been found that the schemes to manage volatile information and to reduce redundancy at storage level can be employed by any crawler.

References

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

[2] Berners-Lee and Daniel Connolly, "Hypertext Markup Language. Internetworking draft", Published on the WWW at http://www.w3.org/hypertext/WWW/MarkUp/HTML.html, 13 Jul 1993.

[3] Junghoo Cho and Hector Garcia-Molina, "The evolution of the Web and implications for an incremental crawler", *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000. Available at http://www-diglib.stanford.edu/cgi-bin/get/SIDL-WP-1999-0129.

[4] Allen Heydon and Mark Najork, "Mercator: A Scalable, Extensible Web Crawler", In Journal of WORLD WIDE WEB , Volume 2, Number 4, 219-229, DOI: 10.1023/A:1019213109274.

[5] Junghoo Cho, “Parallel Crawlers”, In proceedings of WWW2002, Honolulu, hawaii, USA, May 7-11, 2002. ACM 1-58113-449-5/02/005.
[6] A.K.Sharma, J. P. Gupta, D. P. Agarwal, “Augment Hypertext Documents suitable for parallel crawlers”, Proc. of WITSA-2003, a National workshop on Information Technology Services and Applications, Feb’2003, New Delhi.
[7] http://research.compaq.com/SRC/mercator/papers/www/paper.html
[8] Jonathan Vincent, Graham King, Mark Udall, “General Principles of Parallelism in Search Optimization Heuristics”,
[9] RFC 1950 (zlib) ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html.
[10] A.K. Sharma, J. P. Gupta, D. P. Agarwal, “An alternative scheme for generating fingerprints for static documents”, Journal of CSI, Vol. 35 No.1, January-March 2005.
[11] A.K.Sharma, J. P. Gupta, D. P. Agarwal, “A Novel Approach towards Efficient Management of Volatile Information”, Journal of CSI, Vol. 33, No. 3, Mumbai, India, Sept’ 2003.
[12] Robert C. Miller and Krishna Bharat, “SPHINX: a framework for creating personal, site-specificWeb-crawlers”, http://www7.scu.edu.au/programme/fullpapers/1875/com1875.htm
[13] Vladislav Shkapenyuk and Torsten Suel, “Design and Implementation of a High performance Distributed Web Crawler”, Technical Report, Department of Computer and Information Science, Polytechnic University, Brooklyn, July 2001.
[14] Brian Pinkerton, “Finding what people want: Experiences with the web crawler.” In Proceedings of 1st International World Wide Web Conference 1994.
[15] Sergey Brin and Lawrence Page, “The anatomy of large scale hyper textual web search engine”, Proceedings of 7th International World Wide Web Conference, volume 30, Computer Networks and ISDN Systems, pp 107-117, April 1998.
[16] Junghoo Cho and Hector Garcia-Molina, “Incremental crawler and evolution of web”, Technical Report, Department of Computer Science, Stanford University.

[17] 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.
[18] 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 (WWW),* May 2004.