Parallel Crawler Architecture and Web Page Change Detection

DIVAKAR YADAV
AK SHARMA
J.P.GUPTA
Computer Science & Information Technology
Jaypee Institute of Information Technology University
A-10, Sector -62, Noida
INDIA
dsy99@rediffmail.com, ashokkale2@rediffmail.com, jp.gupta@jiit.ac.in

***Abstract*:** In this paper, we put forward a technique for parallel crawling of the web. The World Wide Web today is growing at a phenomenal rate. It has enabled a publishing explosion of useful online information, which has produced the unfortunate side effect of information overload. The size of the web as on February 2007 stands at around 29 billion pages. One of the most important uses of crawling the web is for indexing purposes and keeping web pages up-to-date, later used by search engine to serve the end user queries. The paper puts forward an architecture built on the lines of client server architecture. It discuses a fresh approach for parallel crawling the web using multiple machines and integrates the trivial issues of crawling also. A major part of the web is dynamic and hence, a need arises to constantly update the changed web pages. We have used a three-step algorithm for page refreshment. This checks for whether the structure of a web page has been changed or not, the text content has been altered or whether an image is changed. For The server we have discussed a unique method for distribution of URLs to client machines after determination of their priority index. Also a minor variation to the method of prioritizing URLs on the basis of forward link count has been discussed to accommodate the purpose of frequency of update.

Keywords: Parallel crawler, Change detection, Multi-threaded server, Structural and content changes, Client crawlers.

1. Introduction

1.1 Definition

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 in parallel, downloading web pages independently by browsing the web [2]. These processes need to be coordinated in order to ensure that there is an optimal usage of resources such as bandwidth, memory etc. Many search engines do use some sort of parallelization in their architectures, but a lot of options still remain to be explored. The whole work is divided as follows. In current section we have discussed about the introduction of crawlers and related work done for parallel crawlers and page refreshment techniques. Section 2 discusses the proposed architecture for the parallel crawler. In section 3 we discuss algorithms used for detecting changes in web pages and finally in section 4 we concluded our work along with future directions.

1.2 Overview

In this paper, we propose a new design architecture for building a parallel crawler. The main challenge while making such a design is to maximize the performance of such a crawler and keeping the overheads of memory, bandwidth, etc. which result due to parallelization, to minimum. Each process also needs to work in the most efficient manner to ensure its smooth functioning. For that we need to tackle some common issues faced by the crawling processes. The following issues should be kept in mind while designing a parallel crawler:

If all processes in a parallel crawler are independent of each other, they start from their seed URLs and extract URLs from them and add them to their queues. It is quite possible that different processes download the same pages since two different pages can point to that same page [2]. This leads to wastage of memory and bandwidth. The database cannot have multiple copies of same web page. These copies may be different, as they may have been downloaded at different time instants. Therefore, the quality of the web repository of the search engine suffers.

To combat the dynamic nature of the web, we need to propose a suitable approach which decides when to re-download an existing page based on whether a page is changed or not [3-5, 18-20, 27-28].

The client crawler machines need to crawl URLs decided according to some function or rule. This is done to clearly demarcate each process's area of operation.

Indexing of web pages for search purposes is a major issue. We need to identify the keywords associated with each page [6]. This index is used to locate the exact location of the page and the relevance of that page.

Also other important issues include frequency of page updating, page revisit policies [7-8, 16-17], and prioritizing pages on basis of certain parameters.

1.3 Related Work

Web crawlers have always been an intriguing area of interest since the advent of web. With the growing size of web, a lot of work has also been done before to optimize the performances of web crawlers. Issues addressed by them include the following:

- ***Crawler Architecture:*** Work done describes various architectures under which crawlers of certain current search engines are working. [9] Describes the architecture of the crawling technique used by Google whereas [10] studies the Compaq SRC crawler. Although these papers describe the macro view of the crawler architecture used by them, but little insight or detail has been provided by them regarding the issues related to parallel crawlers that have been discussed above. [2] describes a parallel crawler with multiple architectures along with metrics for evaluation.
- ***Page Update policies:*** Each crawler needs to update the pages on a periodic basis to improve the quality of its databases. [8] discusses scheduling algorithms for crawlers to index the web on a regular basis. [7] describes the various freshness metrics used for gauging the freshness and quality of a local copy of a web page.
- ***Page priority method:*** To prioritize a page over another, certain methods and parameters have been proposed that are also used by modern day search engines [12-14]. These methods take into account various parameters such as link count for a certain page or the keyword occurrence frequency to provide a suitable parameter for page relevance.
- ***Other techniques for crawling*****:** Apart from parallel crawling, other techniques to crawl the web have been proposed which include distributed, incremental, focused etc. [11-13]. All these techniques have their own distinct and significant advantages under different circumstances or constraints. But they may return better results for crawling if used in collaboration with parallel crawlers.

N. Sato et all [29] discussed the problems faced by conventional search engines regarding update intervals. They described that it is difficult for conventional search engines to make their interval short because they are based on centralized architecture. They proposed cooperative search engine, which is based on distributed architecture. In cooperative search engine, a large search engine is constructed with multiple local Meta search engines that cooperate with each other to shorten the update interval.

O. Papapetrau & G. Samaras [26] proposed a location aware method, called IPMicra that utilizes an IP address hierarchy, and allows crawling of links in near optimal location aware manner.

Ntoulas **[21]** collected a historical database for the web by downloading 154 popular Web sites (e.g., acm.org, hp.com and oreilly.com) every week from October 2002 until October 2003, for a total of 51 weeks. The average number of web pages downloaded weekly was 4.4 million. The experiments show that a significant fraction (around 50%) of web pages remain completely unchanged during the entire period they they studied. To measure the degree of change, they compute the shingles of each document and measure the difference of shingles between different versions of web documents. They show that many of the pages that do change undergo only minor changes in their content: even after a whole year, 50% of the changed pages are less than 5% different from their initial version.

Fretterly **[22]** performed a large crawl that downloaded 151 million HTML pages. They then attempted to fetch each of these 151 million HTML pages ten more times over a span of ten weeks during Dec. 2002 to Mar. 2003. For each version of each document, they compute the checksum and shingles to measure the degree of change. The degree of change is categorized into 6 groups: complete change (no common singles), large change (less than 30% common shingles), medium change (30%-70% common shingles), small change (70%-99% common shingles), no text change (100% common shingles), and no change (same checksum). Experiments show that about 76% of all pages fall into the groups of no text change and no change. The percentage for the group of small change is around 16% while the percentage for groups of complete change and large change is only 3%. The above results are very supportive to our studies. They suggest that incremental method may be very effective in updating web indexes, and that searching for new information appearing on the web by retrieving the changes will require a small amount of data processing as compared to the huge size of the Web.

[23] Discuss detecting changes in XML documents. The algorithms discussed in the paper uses signature to match (large) sub trees that were left unchanged between the old and new versions.

L. Francisco-Revilla [24] describe the Walden's Path Path manager, which assists a maintainer in discovering when relevant changes occur to linked resources. They also have emphasized that structural changes are key in determining the overall change and that presentation changes are considered irrelevant. The evolution of change is based on document signatures of the paragraphs, headings, links and keywords. The path manager keep track of original, last valid, and last collected signatures so user can determine both long term and short term changes depending on their particular condition. The author has discovered that the path manager has been designed to work in a distributed environment where connectivity to documents is unpredictable. Its architecture and installation provides users with control over system resources consumed.

Y Wang [25] proposed X-Diff, an effective algorithm that integrates key XML structure characteristics with standard tree to tree correction techniques. Further in the paper they argue that using the unordered tree model is more suitable for most database and web application than using the ordered tree model.

2. Proposed Architecture of The Crawler

The architecture is mainly comprised of 2 main proposed components:

- Multi-threaded Server (MT-Server)
- Client Crawlers

The MT-Server is the main coordinating entity that manages a connection pool with the client machines involved in the actual process of downloading the pages. The server itself does not download any pages.

The client crawlers collectively refer to all the different client machines interacting with the server. The number of clients supported by the design is not fixed. It may vary according to available resources and the scale of actual implementation. Also, there is no inter client communication and all interaction is done between the server and an individual client.

2.1. MT-SERVER

The MT server is the main component of the architecture, involved in interaction with client processes that ensures that there is no need for inter client communication.

2.1.1 URL Redistribution:

The main feature of server is to redistribute the URLs received from different clients to appropriate client machines for their actual download. The server needs to ensure that:

- There is no copy of the URL in the database already. This is important to avoid wasting resources required for crawling and storing web pages.
- Also, at the time of updating a web page, the server should reassign the URL to maintain consistency and avoid overlap overheads caused by parallelization of the crawling process.

For this, the server should use such a method or a function, which returns a consistent value for a given input URL. The URLs may be distributed according to their IP addresses. This is done in order to assign proximate web pages to a client crawler in case of geographically distant crawlers. This reduces network load and download time and saves significant amount of system resources.

Other methods that could be used to aide the server's decision-making processes are conversion of IP addresses to a single value or reducing the URL to a single word/string using a function or method so that

the obtained value can be used for determining the target client machine. eg. IP Number.

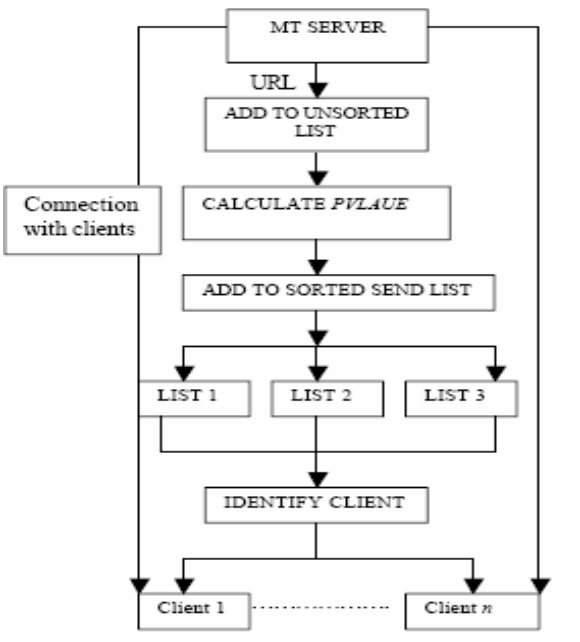


Fig 1. MT server Architecture

2.1.2 Establishing 2-way Communication with Client:

One of the most important tasks performed by the server in the architecture is the establishment and maintenance of 2-way communication with the clients. Following issues should be kept in mind while implementing the server:

- If a client loses connection with the server, the server process should not be disrupted as it will then disrupt the overall crawling process.
- It should be possible to add a new client machine to an ongoing crawling scenario without any temporary halt of the overall system. The server should automatically start the allocation of URLs to that client without any inherent effects on the other client processes or the overall system.
- After the communication channel has been established by the server with each client, the server should send its seed URL to the client which gives the earliest request and then wait for its response. The client's response would be in the form of a list of URLs crawled by it. The server will then be required to redistribute each URL of that list to the suitable clients to carry forward the download process.
- In order to achieve parallelization, the server has to maintain independent threads with each client and carry out the tasks of receiving and sending URLs. The server should ensure that it does not lose the connection with any client and should automatically stop sending the URLs for crawling to clients who have lost connection, as not doing so will result in data loss.
- Once a normal crawling routine has been established, the server should routinely receive lists from clients, sort them by priority and resend each URL to the suitable client.

2.1.3 Relevance Check and Priority Assignment:

After receiving URLs from clients, the server before determining the client machine for assignment of download process must check the URLs received for the following:

- In case the crawler to be built, is a topic specific or a focused crawler, where the quality of the pages to be downloaded is very important, to save on memory and to avoid the downloading of irrelevant web pages, the server should predetermine the relevance by parsing the page contents without downloading and check for the keywords of the page and compare them with the relevant keywords of the main topic or context for which crawling has to be done.
- If a URL is found to be relevant for downloading purpose or in case of full web crawling systems, it has to be sent to the client for actual download. But to maintain the quality of crawling, it is desirable that there should be a method to determine the importance of a page. The following points should be considered while redistributing the URLs.

a) The parameter used to classify URLs in the above categories is the difference of the backward link count (BkLK) and the forward link count (FwdLK) as discussed in [14].

$$Pvalue = af*FwdLK - af*BkLK$$

Where af = aging factor

b) This difference will be known as the pvalue (priority value) of that page. A URL having the difference between FwdLK and BkLK as the highest would be given the highest pvalue.

c) To calculate the values of FwdLK, the server would parse the page without downloading it for just to return the number of links in it.
d) To estimate the number of BkLK, the server would refer to the existing database built by client crawler machines to calculate how many pages refer to this page from the point of view of current database.
e) Initially, the FwdLK will hold higher weight age as there will be no or very few URLs pointing to the current URL as the database would be still in a nascent or a growing stage. But as the database of indexed pages will grow, the weight age of BkLK will gradually increase.
f) Obviously this will result in lower pvalue and that page will hold lower priority. This is important since we do not want to assign high priority to download a page already downloaded and updated very frequently as indicated by the high BkLK value. But we do want to prioritize addition of new pages to our repository of indexed pages, which has a nil BkLK, but high FwdLK as it leads to a higher number of pages.

The server will sort the list of URLs received from the client according to descending order of pvalue. It will then send this sorted list of URLs to the clients in order to maintain quality of crawling. This method is particularly useful as it gives weight age to current database and builds a quality database of indexed web pages even when the focus is to crawl the whole of web. It works well if the database is in growing or in the maturity stage. Also, the method would work well for broken links as such a URL will have a 0 value for FwdLK. Even if it is referenced from pages in the database, pvalue will always be negative resulting in low page priority.

For example:

Unsorted list for sending to clients

URL	FwdLK	BkLK	Pvalue
A	13	8	5
B	22	10	12
C	17	9	8
D	10	11	-1
E	5	3	2

Sorted List for sending to clients:

URL	List No.
A	2
B	1
C	1
D	3
E	3

As the different clients will keep sending their lists to the server, the server will first add them to its list of unsorted URLs, calculate their different parameters and then add them to the sorted list of URLs.

It may be the case that a low FwdLK count but a highly referenced URL may always find itself at the bottom of the sorted queue. One method to get rid of this situation is to break the sorted list into three equal sized lists on the basis of pvalue, i.e. top one-third of the URLs will be sent to list1, middle one-third will be sent to List2 and bottom one-third will be sent to list3. Then redistribute them with a frequency such that for every 4 URLs of list1, 2 URLs of list2 and 1 URL of list3 are sent. This will ensure that no URL is completely ignored. URLs further added to the main sorted list will then be redirected to their respective lists on the basis of pvalue.

2.2 Client Crawlers

The client crawler collectively refers to all the client machines, which are involved in the actual process of crawling. These clients are dependent on the server for receiving the URLs they are supposed to crawl. These clients perform lot of operations other than downloading and storing of web pages. The basic architecture of client crawlers is given in fig 2.

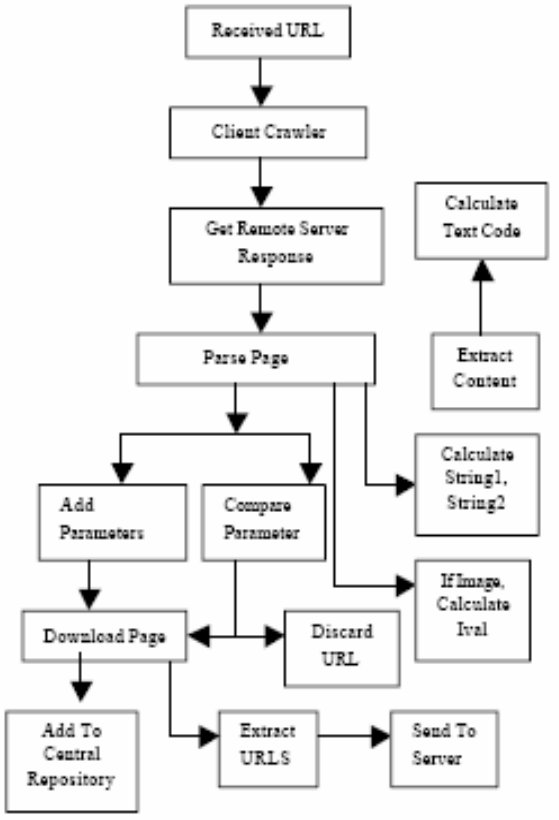


Fig 2. Client Crawlers Architecture

2.2.1 Actual Downloading of Pages:
The client crawler machines are part of the architecture which is mainly concerned with the actual downloading of the web pages. The data repository maintained by our crawling architecture is a central one where all the clients have the right to add pages during their crawling routines.

2.2.2 Parsing Page Content:
Each crawler machine parses the page contents for the following purposes:

- Each page has a certain number of links in it. To maintain the index of BkLK, each link on that page must be stored along with the URL it is appearing in. The client will send to the database the pair of values (link, URL). When the same link reappears on some other page, only the name of URL will be added to the link indexer to the already existing value of link.

For Example: LINK INDEXER

Link	R_URL1
Google.com	Programmingpages.com

- The client also needs to extract the list of all links to be sent to server for their redistribution. The server will give the client machines URLs to be crawled based on the methods discussed in 2.1.1.
- The main motive behind parsing all the contents in the page is to extract the page updating parameters that will be used to check whether a page has been changed or not. These parameters are discussed in detail later.

The client crawler machine should be robust enough to be able to handle all types of web pages appearing on the web. With growing technology, the web is no longer limited to simple HTML pages, but it consists of a whole variety of pages used to display dynamic content and ever changing layouts. The client should be implemented in such a manner that is able to accurately parse the web page content and handle each type of page in an efficient manner. The client should be robust enough to handle the pages, which do not allow themselves to be crawled [15]. Also it should automatically discard URLs that are referenced but do not exist any more on the net.
Therefore, a client machine before starting the actual download of a page should check for its actual existence on the web using the remote server response. If there is no response from the server or the page is forbidden to be visited from certain networks, it should immediately discard it without waiting for further responses in order to conserve resources for faster crawling.

3. Proposed Page Update Algorithms

About 60% of the content on the web is dynamic [16]. It is quiet possible that after downloading a particular web page, the local copy of the page residing in the repository of the web pages becomes obsolete compared to the copy on the web. Therefore a need arises to update the database of web pages. Once a decision has been taken to update the pages, it should be ensured that minimal resources are used in the process. Updating only those elements of the database, which have actually undergone a change, can do this. Importance of web pages to be downloaded has been discussed in the above section. It also checks whether the page is already there in the database or not and lowers its priority value if it is referred rather frequently.
In this section, we discuss some algorithms to derive certain parameters, which can help in deriving the fact whether the page has changed, or not.
These parameters will be calculated at the time of page parsing. When the client again counters the same URL, it just calculates the code by parsing the page without downloading the page and compares it to the current parameters.
If changes in parameters are detected, it is concluded that the page has changed and needs to be downloaded again. Otherwise the URL is discarded immediately without further processing.
The following changes are of importance when considering changes in a web page:

- Change in page structure.
- Change in text contents.
- Change in image (Hyperlinked or as a part of the page).

All the above steps are not necessary to be taken care of. A parameter is compared only if the preceding parameter returns no change.

3.1 Changes in Page Structure.

Here we propose two algorithms for detecting changes at structural level. Each have its own advantage but first is better in respect to memory required for maintaining the data which we get after performing the algorithm on web pages.

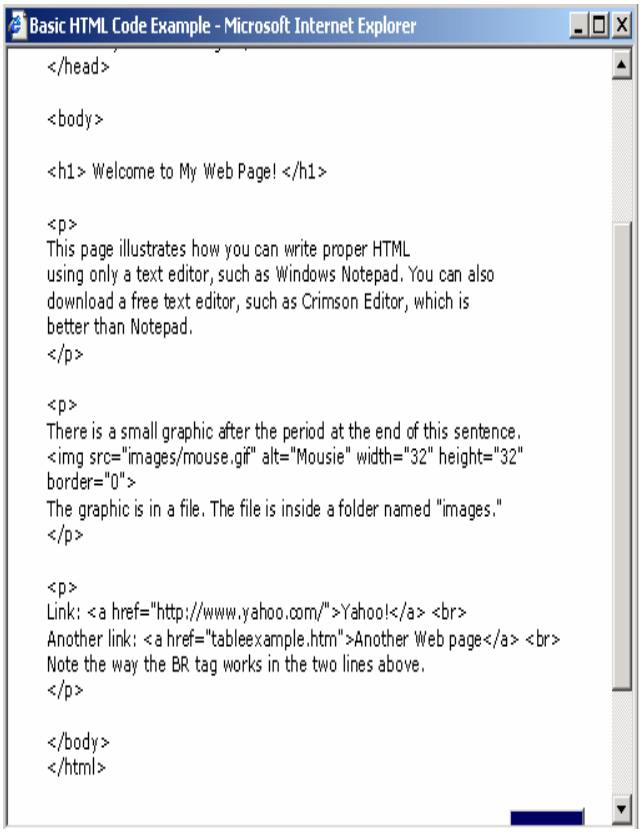
3.1.1 I-Method

This method tries to capture the changes in the structure of pages. Here by structure we mean how are the different texts and images and other objects displayed on that page. All these objects are designed using HTML or other formatting tools. These tools use tags to define their characteristics and actual appearance on the page. Any change in structure will lead to rearrangement of tags.

The algorithm creates two strings using the tags of that page.

- The first string will store the characters appearing at first position in the tag for all tags in the order they appear in web page.
- The second string stores the character at the last position in a tag for all tags in the order they appear in web page.
- In case the tag contains only one letter, it will be repeated as it is, in the second string too.

For example consider the following page:



For above web page following strings are formed.

String1: [hhtbhppp]

String2: [ldey1ppp]

The proposed method offers the following advantages:

- The traditional approaches of storing the pages as a tree structure uses a lot of storage space as well as causes a lot of inconvenience at time of refresh, as the tree structure has to be compared. Here only a string has to be compared to perform the desired operation.
- Even if any tag is added or deleted from the page, the new string will record the change and the difference will show in the comparison.
- At the time of page updating, the client crawler only needs to check these two strings to determine changes.
- If the String1 itself returns a changed value, there is no need for comparison with the second string or with other change parameters for that matter.
- This method will work for different pages with different and varied formatting styles accurately capturing their structures at a given point of time.
- For most cases, a check with the first string should suffice. The second string check is included just to add surety to the method as the check for String1 may fail in the unlikely case of a tag starting with some letter replacing another tag starting with the same letter.

3.1.2 II-Method

The change extractor based on this algorithm includes two phases: document tree construction and level order child enumeration (tree traversal/parsing by level order).

The structure of every node of the tree representing the web page shall contain the following information's

- **ID:** this index stores the unique id for each node of the tree.
- **CHILD:** This index stores the information about children of each node.
- **LEVEL:** This index stores the levels, where the node exists in the tree.
- **CONTENT VALUE:** this index stores the Root Mean Square (RMS) of sum of ASCII values of characters.

For example if we get tree structure of a particular web page as is shown in Fig. 3a and latter on some change occur in web page so that we get tree structure as is shown in Fig.3b following tables using BFS are created.

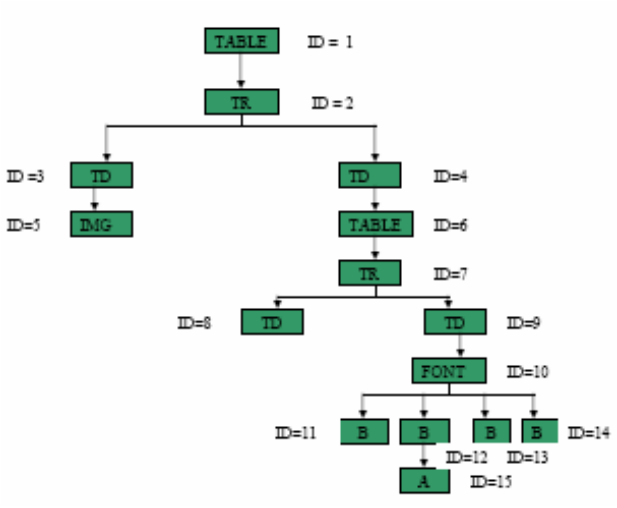


Fig. 3a Initial Structure (before addition)

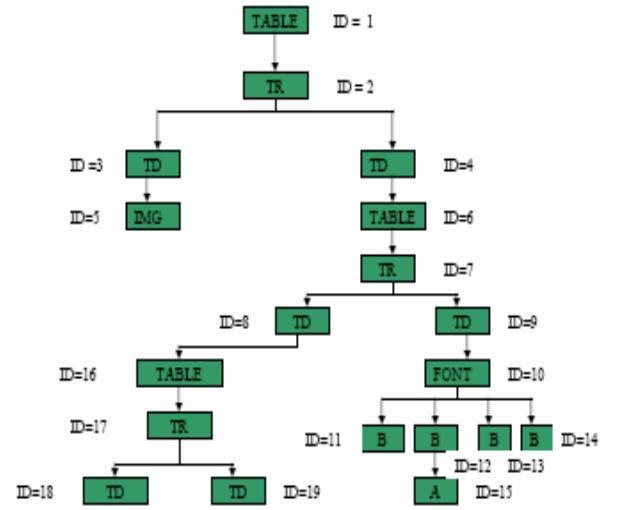


Fig. 3b Modified Structure (after addition of nodes

Table-I Level structure using BFS for above initial and modified tree structure of the web page

LEVEL	INITIALLY	LATTER
Level-1	1	1
Level-2	1	1
Level-3	2	2
Level-4	2	2
Level-5	1	1
Level-6	2	2
Level-7	1	2
Level-8	4	5
Level-9	1	3

Before changes occur (Fig. 3a)

Level = {1 2 3 3 4 4 5 6 6 7 8 8 8 8 9}
ID = {1 2 3 4 5 6 7 8 9 10 11 12 13 14 15}
CHILD ={1 2 1 1 null 1 2 null 1 4 null 1 null null null}
Level_array ={1 1 2 2 2 2 1 2 2 1 4 4 4 4 }

After changes have occured (Fig. 3b)

Level = {1 2 3 3 4 4 5 6 6 7 7 8 8 8 8 8 9 9 9}
ID = {1 2 3 4 5 6 7 8 9 16 10 17 11 12 13 14 18 19 15}
CHILD ={1 2 1 1 null 1 2 1 1 1 4 1 nul 1 null null null null null }
Level_array={1 1 2 2 2 2 1 2 2 2 2 5 5 5 5 5 3 3 3 }

Now, by comparing the two sets we get the idea that the modification has been done at LEVEL 7, 8 and 9. In order to find the modification at a given level, we use the level order traversal. The algorithm for level order traversal with breadth first search will give us the location where the change has taken place.

3.2 Changes in Text Content

This is the next step to be carried out if the first level discussed above for determining changes does not find any changes. It may be the case that the structure of the page remains intact but there are some definite changes in the actual text content of the page. These changes won't be captured by the above method.
Here also we are proposing two techniques to identify the changes at content level

3.2.1 I-Method

In this method, we assign a code to all text content appearing in a web page. At the time of page updating, only comparison will be made to the text code of that page and if any change in that value is detected for the actual copy on the web as compared to the local copy, the page will be refreshed or re-crawled.
The formula for text coding is as follows:

$$\frac{\sum(frequency) * ASCII_{symbol}}{\text{Distinct symbol count}}$$

For example, consider the following partial text content from
w*ww.msnbc.msn.com/id/17662246/site/newsweek*

March 26, 2007 issue - The stereotype of the "dumb jock" has never sounded right to Charles Hillman. A jock himself, he plays hockey four times a week, but when he isn't body-checking his opponents on the ice, he's giving his mind a comparable workout in his neuroscience and kinesiology lab at the University of Illinois. Nearly every semester in his classroom, he says, students on the women's cross-country team set the curve on his exams. So recently he started wondering if there was a vital and overlooked link between brawn and brains—if long hours at the gym could somehow build up not just muscles, but minds.

For above webpage's text contents following calculations are made:
The ASCII sum of characters: **56652**
Total character count: **619**
Distinct character count: **43**
Therefore, the code for the text will be: **1317.488403**

Even minute changes in above text, results in significant changes in parameters as given below. Words in bold are changed one.

*March 26, 2007 issue - The stereotype of the "dumb jock" has never sounded right to Charles Hillman. A jock himself, **he used to play** hockey **five** times a week, but when he isn't body-checking his opponents on the ice, he's giving his mind a comparable workout in his neuroscience and kinesiology lab at the University of **Florida**. Nearly every semester in his classroom, he says, students on the women's cross-country team set the curve on his exams. So recently he started wondering if there was a vital and overlooked link between brawn and brains—if long hours at the gym could somehow build up not just muscles, but minds.*

Again we perform parameter calculations as below for above changed webpage's contents.
The parameters are as follows:
The ASCII sum of characters: **57113**
Total character count: **625**
Distinct character count: **43**
Therefore, the code for the text will be: **1328.209351**

3.2.2 II-Method

This is similar to I-method except that here we use different formula to calculate the code (RMS) for webpage's content. The formula for text code calculation is as follows:

$$R.M.S = \left[\frac{(a1)^2 + (a2)^2 + \ldots\ldots\ldots\ldots + (a'n)^2}{n} \right]^{1/2}$$

Where a_i is ASSCII code of ith character.

Example:
Initial Content
Nearly every semester in his classroom, he says, students on the women's cross-country team set the curve on his exams. So recently he started wondering if there was a vital and overlooked link between brawn and brains—if long hours at the gym could somehow build up not just muscles, but minds.

For above webpage's text contents following calculations are made:
Sum of square of ASCII codes of char = 2830106
Total Char = 296
Code for texts = 5.683418

Changed Content
*Nearly every semester in **her** classroom, **she** says, students on the women's cross-country team set the curve on her exams. So recently **she** started **speculating** if there was a vital and overlooked link between brawn and brains, if long hours at the gym could somehow build up not just muscles, but minds.*
Again we perform parameter calculations as below for above changed webpage's contents.
Sum of square of ASCII codes of char = 2876660
Total Char = 300
Code for texts = 5.653573

The proposed methods offer the following advantages:
- It gives a unique code for all text contents on a particular page.
- The formulae are designed to be such that even a minute change of addition or deletion of a single blank space is recorded significantly and with pin point accuracy by this code.
- The use of coding of text system eliminates the need for word by word parsing and comparison of each word at the time of page updating.

- Only the code of that page is compared, there is no issue of storing the whole page as an indexed structure, hence saving on large amount of storage.
- ASCII values have been used in the formula because each symbol has a distinct representation in ASCII table leading to no ambiguity.

3.3. Change in Image

While the above two methods may suffice while dealing with normal pages which uses tags for formatting their structure and text contents, they will fail for image links. In this section we propose a method to derive a code for images to determine whether they have undergone a change or not. Ideally a change in a link to an image hyperlink will be reflected in the label of the hyperlink for that image and the same will be depicted by the formula proposed above. But in case the text does not change but the image is replaced, it will still be left undetected. We propose the following method for image change detection:

- The first step requires the image to be scaled to a standard size of n*n. Here n is of the form 2^x, where value of x may vary from 4 to 6.
- Convert the image to two tone and read as an n*n array with each value being either 0 or 1.
- For each row of n elements, we will take 2^4 elements at a time and convert it to a 4 digit hexadecimal number. This will result in each row being converted to $n/2^4$ elements from n elements with each element being a 4 digit hexadecimal number.
- After doing the same operation on all the rows, we obtain an $n/2^4 * n/2^4$ array.
- Simply, by computing the determinant of such an array, we can reduce the image to as single value (ival).
- For each image, an ival will be stored and at time of page updating, the client machines without actual download of image will calculate this ival.

4. Conclusion and Future Work

The architecture that has been proposed by us in this paper has the following distinct advantages:

- The issue of overlap of downloads by client crawlers are addressed, as the main server does not give the same URL to different clients who themselves check whether it is already existing or not.
- The centralized database of downloaded URLs reduces the dependency of the system on a single client.
- The architecture is easily scalable. This is unlike the scenario discussed in [2], where adding more machines to the crawling process would add to overlap overheads.
- The algorithm for checking page update parameters is designed to show even the smallest of change in a web page and leaves no ambiguity as the values stored for the parameters are distinct for small details as well. This helps the client crawler to clearly determine whether or not a page has changed and saves memory and bandwidth overheads.

Potential work related to the parallel crawler that may be added on, is the adaptation of this model for building a full fledged search engine. The algorithms suggested for detecting web page changes, if incorporated in search engines along with parallel crawler model will help keeping the web pages refreshed.

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] Ling Liu Carlton Pu, Wei Tang**, "**WebCQ – Detecting and Delivering Information Changes on the Web". Proceedings of the ninth international conference on Information and knowledge management McLean, Virginia, United States 2000. Page(s): 512 – 519.

[4] Daniel Rocco, David Buttler, Ling Liu, "Page Digest for Large-Scale Web Services", E-Commerce2003. (CEC'03), IEEE International Conference on. 24-27 June 2003 Page(s):381 - 390.

[5] David Buttler, Daniel Rocco, Ling Liu, "Efficient Web Change Monitoring with Page Digest"**,** Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, New York. Page(s): 476 – 477.

[6] Monica Peshave, Kamyar Dezhgosha, "How Search Engines Work and a Web Crawler

Application". Department of Computer Science, University of Illinois, Springfield USA.

[7] Junghoo Cho and Hector Garcia-Molina, "Synchronizing a database to improve freshness**,** submitted for publication". Proceedings of the 2000 ACM SIGMOD international conference on Management of data. Volume 29 Issue 2. Page(s): 117 – 128.

[8] E. Co.man, Jr., Z. Liu, and R. R. Weber, "Optimal robot scheduling for web search engines". Proceedings of the 11th international conference on World Wide Web WWW '02 Honolulu, Hawaii, USA. ACM Press. Page(s): 136 – 147.

[9] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", In *Proceedings of the Seventh World-Wide Web Conference*, 1998.

[10] A. Heydon and M. Najork, "Mercator: A scalable, extensible web crawler. *Word Wide Web"*, December 1999. Page(s):219–229.

[11] Dustin Boswell, "Distributed High-performance Web Crawlers:A Survey of the State of the Art".

[12] M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs", In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.

[13] S. Chakrabarti, M. van den Berg, and B. Dom, "Focused crawling: A new approach to topic-specific web resource discovery", In *The 8th International World Wide Web Conference*, 1999.

[14] Junghoo Cho, Hector Garcia-Molina, and Lawrence**, "**Efficient crawling through URL ordering Page", In Proceedings of the 7th World-Wide Web Conference, 1998, page(s):161-171.

[15] Robots exclusion protocol. ttp://info.webcrawler. com/mak/projects/robots/exclusion.html.

[16] Junghoo Cho, Los Angeles, Hector Garcia-Molina, " Effective Page Refresh Policies for Web Crawlers", ACM Transactions on Database Systems (TODS). Volume 28, Issue 4 (December 2003). Page(s): 390 – 426.

[17] Junghoo Cho, Los Angeles and Hector Garcia-Molina, **"**Estimating Frequency of Change", ACM Transaction on Internet Technology, Vol 9, No3 , Aug 2003, page(s): 256-290.

[18] Luis Francisco-Revilla, Frank M. Shipman III, Richard Furuta, Unmil Karadkar, Avital Arora, "Perception of Content, Structure, and Presentation Changes in Web-based Hypertext", Proceedings of the twelfth ACM conference on Hypertext and Hypermedia 2001**.** Pages: 205 – 214

[19] Latifur Khan, Lei Wang and Yan Rao, "Change Detection of XML Documents Using Signatures", Proceedings of the 2005 ACM/IEEE conference on Supercomputing SC '05. Page: 69.

[20] Shuohao Zhang, Curtis Dyreson, and Richard T. Snodgrass Schema- Less, "Semantics-Based Change Detection for XML Detection", Washington State University, Pullman, Washington, U.S.A. WISE 2004, Springer-Verlag Berlin Heidelberg 2004 , page(s): 279–290.

[21] A. Ntoulas, J. Cho, and C. Olston. "What's new on the web? The evolution of the web from a search engine perspective", In Proc. 13th International World Wide Web Conference, 2004. Page(s): 1 – 12.

[22] D. Fretterly, M. Manasse, M. Najork, and J. Wiener, "A large-scale study of the evolution of web pages", In Proc. 12th International World Wide Web Conference, Budapest, Hungary 2003. Page(s): 669 - 678.

[23] G.Cobena, S. Abiteboul, A. Marian, "Detecting Changes in XML Documents", Data Engineering 2002 Proceeding 18th International Conference on, San Jose USA. Page(s): 41-52.

[24] L. Francisco-Revilla, F. Shipman, R.Furuta, U. Karadkar, A. Arora, "Managing Change on the Web", Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries 2001. Page(s) 67 - 76

[25] Y. Wang, D.J. DeWitt, J-Y Cai, "X-Diff: An Effective Change Detection Algorithm for XML Documents", Data Engineering, 2003. Proceedings. 19th International Conference on**.** Page(s):519-530

[26] Odysseas Papapetrou, George Samaras "Distributed Location Aware Web Crawling", Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters,NY USA 2004. Page(s):468-470.

[27] D.Yadav, A.K. Sharma and J.P.Gupta, "Change Detection in Web pages", IEEE Proceeding of 10th International Conference on IT, Dec 17-20, 07, Rourkela (India). ISBN: 0-7695-3068-0, Page(s) 265-270

[28] D.Yadav, A.K.Sharma and J.P.Gupta, "Architecture for parallel crawler and algorithm for web page change detection", IEEE Proceeding of 10th International Conference on IT, Dec 17-20, 07, Rourkela (India). ISBN: 0-7695-3068-0 Page(s) 258-264

[29] N. Sato, M. Uehara, Y.Sakai, H. Mori, "Distributed Information Retrieval by Using Cooperative Meta Search Engines", Distributed Computing Systems Workshop, 2001 International Conference on**.** Page(s) :345:350.