I.J. Information Technology and Computer Science, **2015, 06, 59-65**
Published Online May 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2015.06.08

Clustered Webbase: A Repository of Web Pages Based on Top Level Domain

Geeta Rani
Shobhit University, Meerut, 250110, India
Email:singh.geeta0088@gmail.com

Dr. Nidhi Tyagi
Shobhit University, Meerut, 250110, India
Email: mnidhity@rediffmail.com

Abstract — The World Wide Web is a huge source of hyperlinked information; it is growing every moment in context of web documents. So it has become an enormous challenge to manage the local repository (storage module of search engine) for to handling the web documents efficiently that leads to less access time of web documents and proper utilization of available resources. This research paper proposes an architecture of search engine with the clustered repository, organized in a better manner to make task easy for user to retrieving the web pages in reasonable amount of time. The research focuses on coordinator module which not only indexes the documents but also uses compression technique to increase the storage capacity of repository.

Index Terms— URL, URI, TLD.

I. Introduction

In an effort to keep up with the tremendous growth of the World Wide Web, many research projects were targeted on how to organize such information in a way that will make it easier for the end users to retrieve information efficiently and accurately. Information on the web is present in the form of text documents (formatted in HTML) and search engines [1] act as a bridge between end users and web pages. Without search engines, the unlimited source of information stored in web pages remain hidden for users. Search engines include various components for simplifying the task of searching. A search engine is a searchable module which collects information from web pages on the internet, indexes the information and then stores the result in a huge database that is called repository where from it, information can be quickly searched. The main module of search engine is Crawler. A web crawler [2] is a program that retrieves web pages. A web crawler starts by placing an initial set of URLs in a seed queue. Uniform Resource Locator (URL) is a Uniform Resource Identifier (URI), for example http://www.w3.org/default.aspx, http [3] is a protocol with the lightness and speed necessary for a distributed collaborative hypermedia information system and .org is the top level domain , each web page belongs to any one of top level domain like .com, .edu etc. A normal top-level domain (TLD) is one of the domains at the highest level in the hierarchical Domain Name System of the Internet. The web crawler gets a URL from the seed queue, downloads the web page, extracts any URLs in the downloaded page, puts the new URLs in the seed queue, and gets the next URL from the seed queue. The storage module [4] (generally indexer) receives web pages from a crawler, indexes and stores these web pages to the repository. A web repository [4] stores a large collection of 'data objects' referred as web pages. Repository contains the full HTML of every web page or repository is the local database of web pages that are crawled by the crawler. The web pages are compressed in the repository uses various methods of data compression can be used e.g. gzip, zlib. The choice of compression technique is a tradeoff between speed and compression ratio. In the repository, the documents are stored sequentially and are prefixed by doc ID, length and URL.

The paper is organized in six sections. Section 2, discusses about the related work, section 3 highlights the problem domain. The proposed architecture along with various components discussion and illustrative example is discussed in section 4 followed by experimental results and conclusion in section 5 and 6 respectively.

II. Related Work

Web crawler is the major component of the web. Since the invention of the web many crawlers have been introduced. The first web crawler Wanderer [5], followed by Internet Archive [6] crawler that uses multiple machines to crawl the web and find duplicate URLs. Web crawling techniques [2] help to improve the data retrieval process in effective manner. Based on different web crawling techniques different crawlers have been introduced. Parallel crawler, Distributive crawler, focused crawler[7], hidden web [8] are the crawler that each crawler has its own technique to crawl the web.

The available literature has not discussed the storing process of document in the repository. Webbase [9] an experimental web repository, discussed the repository preparation in detail with storing as well as searching process. Webbase is main source of motivation towards the objective of this paper. A cluster balancing approach for efficient storage of web documents proposed an architecture for domain specific cluster based repository.

This research paper proposed architecture of crawler with enhanced repository technique [13] which deals efficiently with the problem of limited size of storage resources and searching complexity. As a web page belong to any one top level domain (TLD) available on internet like .org, .com, .co etc. The memory is partitioned into numbers of variable size blocks and these blocks are called as *clusters* (figure 1) and the partition of memory into blocks is done on the foundation of different domains (.com, .edu, .co etc.) that belongs to web. As the memory blocks creation is based on specific top level domain and there users on web so these clusters are also called as *domain specific clusters*.

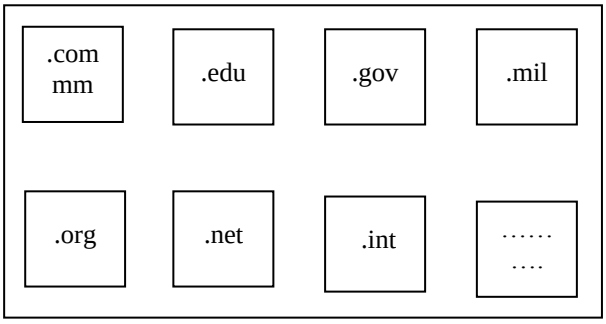

Fig. 1. Domain Specific Clusters

On the other side, indexing is also an important point to be discussed. Basically crawlers uses inverted index for indexing web documents. As a web can have more than one versions of same page and the pages on the web also become obsolete. In this direction different techniques have been proposed for improving index process. Compact full-text indexing of versioned document collections [10] proposes new techniques for organizing and compressing inverted index structures for such versioned document. Efficient phrase-based document indexing for web document clustering [11] presents two key parts of successful document clustering. The first part is a novel phrase-based document index model, the document index graph, which allows for incremental construction of a phrase-based index of the document set with an emphasis on efficiency, rather than relying on single-term indexes only. It provides efficient phrase matching that is used to judge the similarity between documents. The second part is an incremental document clustering algorithm based on maximizing the tightness of clusters by carefully watching the pair-wise document similarity distribution inside clusters. The combination of these two components creates an underlying model for robust and accurate document similarity calculation that leads to much improved results in web document clustering over traditional methods. An improved indexing mechanism to index web documents [12] presents contextual based indexing considers the senses of the keyword for preparation of index of web documents. Huffman coding is an entropy_encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.

III. Problem Identification

The critical look at the available literature reveals that the following points should be considered to develop a more efficient system.

1) *Scalability* – Web is growing every instant and this growth of web challenges the repository to adopt large number of new web pages at every instant with its limited size.
2) *Large updates* - Web changes every moment. Therefore, the repository needs to maintain high rate of modification. These changes occur in the form of web pages that is new version of web pages are added to web so the space occupied by old version must be reclaimed.
3) *Obsolete pages* – At one side pages are added to web and at the other side pages are deleted from website but repository is not notified. So the must be mechanism to detect such obsolete pages and make space utilization for new updates.
4) *Searching* – As repository is local collection web pages and searching within the local collection must lead to retrieve the web pages in the reasonable amount of time.

IV. Proposed Work

A. Architecture

Figure 2 depicts the architecture of the clustered webbase system in terms of main functional modules and their interaction. There are mainly eight modules in architecture –

- Downloader
- Crawler
- Coordinator
- Ranker
- Indexer
- Repository

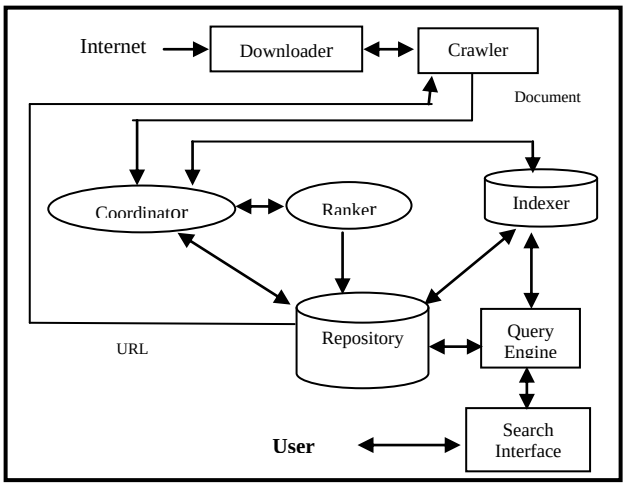

Fig. 2. Proposed Architecture

The associations among these modules represent exchange of information as indicated by directions.

B. Description of the various modules

1) Crawler Module

The working of the crawl module involves the following steps:

i) Fetch URL
ii) Extract URLs
iii) Analyze URL
iv) Forward URL

On receiving the seed URL the crawler forwards it for downloading the Web pages to the downloader. Downloader forward the downloaded page to crawler then crawler extracts URLs and analyze whether the extracted URLs are in syntax error and check duplicate URLs.

2) Downloader

This component takes a URL from URL collector and downloads the corresponding webpage to store it in the local repository.

3) Indexer

Indexer extracts all the uncommon words from each page and records the URL where each word has occurred. The result is stored in a large table containing URLs; pointing to pages in the repository where a given word occurs. This complete process is termed as indexing process. The indexer supports inverted indexing technique. The structure of indexer has four basic Terms:

- keyword
- Domain Name
- Frequency
- Document Names

Table 1. Structure of index

Keyword	Domain Name (Cluster)	Document Name	Frequency
Crawler	.org	Document 1, Document 3, Document 4, Document 7	9
	.com	Document 2, Document 12	
	.co	Document 6, Document 8, Document 10	
Network	.in	Document 11, Document 5	3
	.edu	Document 9	
……	…..	……..	…..

4) Repository

Web repository is a storage module that stores and manages a large collection of 'data objects' in this case web page. So a web repository is basically collection of web pages or web documents. The repository receives web pages from a *crawler*, which is the component responsible for mechanically finding new or modified pages on the web. Repository contains the full HTML of every web page. A web repository need to provide the following functionalities:

- Retrieval of large quantities of data from the Web (page addition)
- Manage meta-data about Web resources:
- Efficient access to stored data

Web repository follows simple coherency model that is store-ones-read-many model for web pages. A web page once stored need not to be changed until page is updated to web. This assumption simplifies data coherency issues and enables high throughput data access. A web crawler application fits perfectly with this model. At the same time, the repository offers applications an access interface (API) so that they may efficiently access large numbers of up-to-date web pages.

5) Coordinator Module

The most important component of the architecture, Coordinator Module receives document from crawler as parameter and performs the following steps:

i) Identify document domain and format
ii) Decide memory block
iii) Compress the document
iv) Update Index
v) Store compressed document

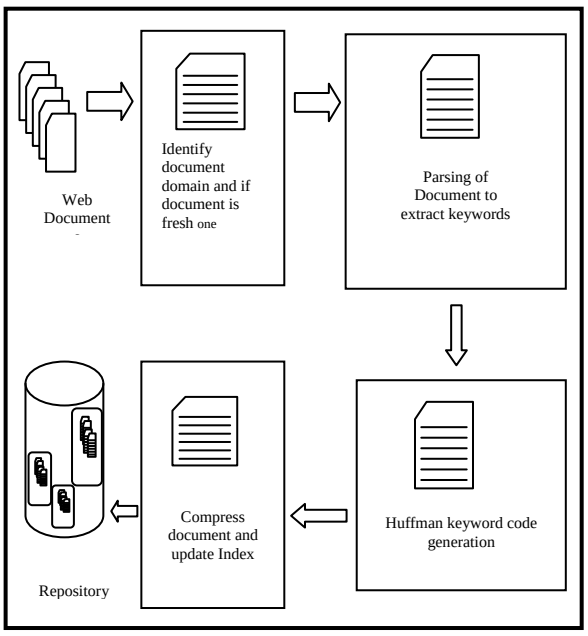

Fig. 3. Working of coordinator module

On receiving the document from the crawler, coordinator module identifies the format and domain of downloaded document. On the basis of domain of that document it decides the cluster to check whether the document is already downloaded or not using URL identifier. Thus reducing the search complexity because

searching is done only in a specific cluster not in linear fashion to complete repository. If the document is fresh one then, coordinator module checks the status of cluster that the cluster can acquire new document or not. If the status of cluster is able to have new document then extract URLs, add them to seed of URLs, the document is parsed and the keywords are identified along with the frequency. Coordinator receives frequency from indexer to generate the Huffman codes using Huffman algorithm. Table 1 shows structure of indexer. In the proposed architecture, frequency depends on the presence of keyword to different documents. The keyword that is present in more number of documents will have higher frequency that leads to less length size Huffman code according to Huffman algorithm. Finally the index is updated and the document is saved in compressed form or else the status of cluster is not able to accommodate new document than less ranked document will be deleted to store new document. Figure 3 depicts how the coordinator module deals with web pages and stores them to repository in compressed form using Huffman algorithm.

In the case when document already exists then ranking module decide the importance of the document. Algorithm of Coordinator module is shown in figure 4.

```
Coordinator_module(domain,document)

{
    Steps:
1.Identify the document domain like doc, html, or pdf etc.;
2.Check the domain of downloaded document , decide the memory block and search it only its specific domain cluster (memory block)whether the document has already downloaded or not;
3.If (document is fresh one)
    {
        3.1 extract the links or references to the other cites from that documents;
        3.2 Compress the document using Huffman algorithm;
        3.3 Update Index;
        3.4 Check the status of decided cluster and if there is a need to create space for new downloaded document then remove less ranked document;
        3.5Store compressed document it into repository;
    }
4. else
    {
        4.1 Call ranking_module (cluster, document);
        4.2 Convert the URL links into their absolute URL equivalents;
        4.3 Add the URLs to set of seed URLs;
    }
}
```

Fig. 4. Algorithm for Coordinator module

6) Ranking module

This component retrieves cluster and document from coordinator module, and increases the rank of document based on popularity to maintain the freshness of repository also help coordinator module to maintain the size of clusters. It calculates rank of document on the basis of popularity, increase rank and return rank. The main steps of Ranker module are shown in figure 5.

```
Ranking_module(cluster,document)
{
    Steps:
    1.Calculate rank of document on the basis of popularity;
    2.Increase rank;
    3.Return rank;
}
```

Fig. 5. Algorithm for Ranking module

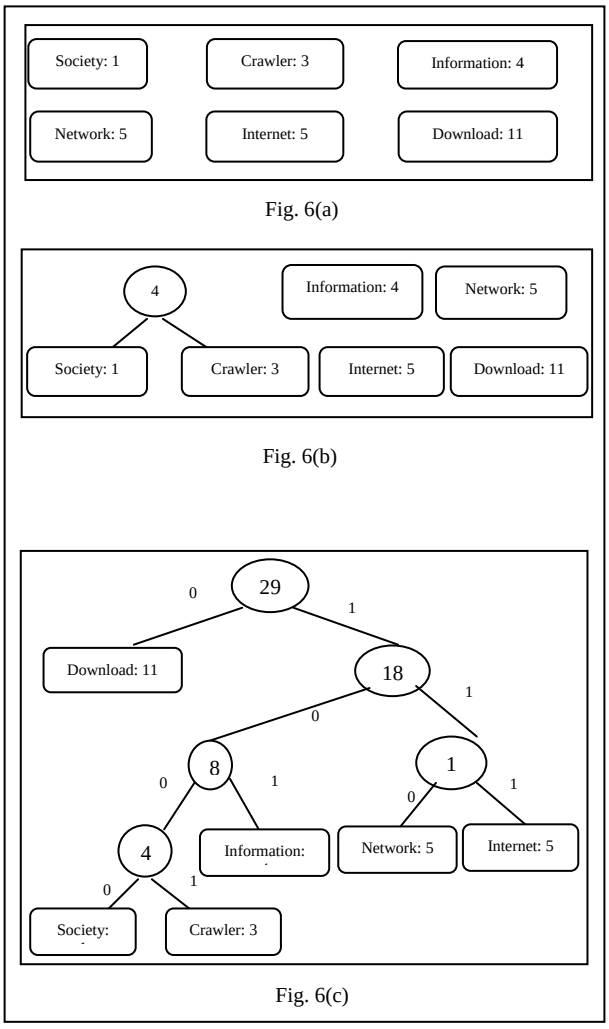

Fig. 6(a)

Fig. 6(b)

Fig. 6(c)

Fig. 6. Tree creation

Example:

Step (i) Parse the document and collect the unique words.

Unique Words: Internet, Crawler, Information, Network, Society, Download

Step (ii) Check indexer to find the frequency of existing words and the frequency of new keyword is assigned as zero.

Step (iii) Increment each frequency by one and arrange them in increasing order, as shown in Figure (a).

Step (iv) Huffman tree is created using Huffman algorithm. Figure (b) and (c) shows different levels of tree creation.

Step (v) Mapping of Huffman codes to keywords after applying Huffman's algorithm as shown in figure 6.

Step (vi) File is stored in compressed form using variable size Huffman codes

Table 2. mapping of keyword to Huffman codes

Serial no	Keyword	Huffman Code
1.	Download	1
2.	Information	101
3.	Internet	111
4.	Network	110
5.	Crawler	1001
6.	Society	1000

V. Implementation and Results

Demo of Crawler Module is implemented in java starts with crawling process with a set of seed URLs. The crawler module downloads web pages from the web. Java Tika application is used for parsing the document which are downloaded by Crawler module. By parsing the downloaded documents, all the keywords are extracted and a text file is created with all keywords. This text file is submitted to the Coordinator module. Coordinator module of proposed system is implemented in object oriented programming (C++) language. Module read contents of text file and assigns frequency. In the proposed architecture, the frequency assigns to the keywords the total no of documents in repository in which the keyword is present but here is an assumption for assigning the frequencies to keywords. The frequency of keywords is assigned by using random function.

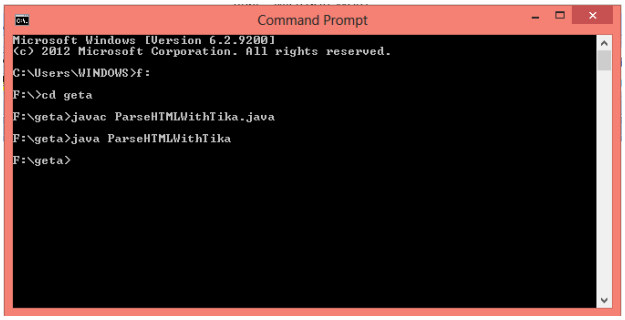

Fig. 7. Parsing of document

After the assignment of frequency, Huffman compression is applied by generating Huffman tree and Huffman codes are created for keywords and finally the page is stored using Huffman codes. At the end there is a equality test is conducted to compare the size of actual file and compressed file.

1) Parsing of document: Tika application parses the downloaded document. Figure 7 shows the parsing process of document.

```
for(int i=0;i<listOfFiles.length;i++)
{
        if(listOfFiles[i].isFile())
        {
    is = new FileInputStream(listOfFiles[i]);
   ContentHandler contenthandler = new
BodyContentHandler();
   Metadata metadata = new Metadata();
   Parser parser = new AutoDetectParser();
   parser.parse(is, contenthandler, metadata, new
ParseContext());
     String str=contenthandler.toString().trim();
   String[] result = str.split("\\s");
      for (int x=0; x<result.length; x++) {

     fetched_token.add(result[x]);

      }
        }
}
```

Fig. 8. Implementation Code for parsing of document

Figure 8 represents keyword fetching process by Tika application.

2) Frequency Assignment: Co-ordinator module assigns frequency to all the keywords using Random() function. . Figure 9 shows frequency assignment process.

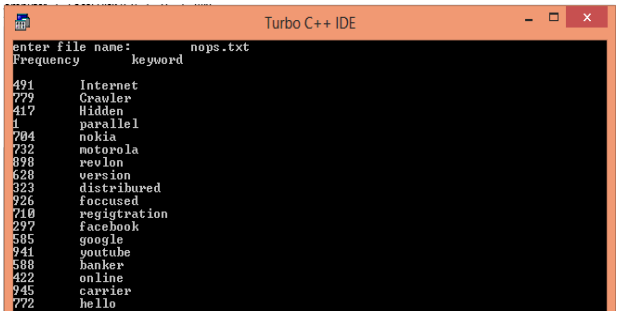

Fig. 9. Processing of document(assignment of frequency to keywords)

Here figure 10 shows how Random() Function assigns a random number to each keyword.

```
ifile.open(file_name1,ios::in);
while(ifile)
{
        Keyword=ifile.getline();
         x=random(1000);
        add. Node();

 }
ifile.close();
```

Fig. 10. Implementation Code assignment of frequency to keywords

3) Huffman Tree creation:Huffman tree is created assuming each keyword as a character. Figure 11 represents Huffman tree nodes values.

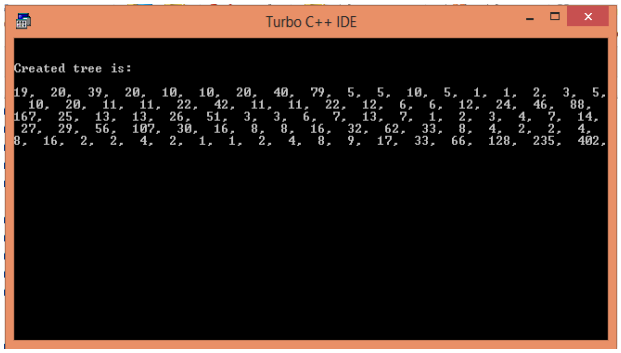

Fig. 11. Processing of document(tree creation)

Figure 12 showing steps of Huffman tree creation by adding two smallest frequencies and creating a node until all the frequencies are not added.

```
void Huff_man::Tree_create(P_queue & Q)
{
Node *ptr1,*ptr2;
while((Q.Q_front->link)->link!=NULL)
{
      ptr1=Q.del();  ptr2=Q.del();
      Node* New=new Node;
      New->frq=ptr1->frq+ptr2->frq;
      New->RC=ptr2;
      New->LC=ptr1;
      Q.insert(New);
 }
      ptr1=Q.del();  ptr2=Q.del();
      Node* New=new Node;
      New->frq=ptr1->frq+ptr2->frq;
      New->RC=ptr2;
      New->LC=ptr1;
      ptr1=ptr2=NULL;
      Tree_head=New;
      cout<<"\n\nCreated tree is:\n\n";
      Tree_show(Tree_head);
       return;
}
```

Fig. 12. Implementation Code for creation of tree

4) *Compresed file* : Finally the document is compressed using variable length huffman code. Figure 13 represents file with variable length Huffman code.

Fig. 13. Processing of document(Compresed file)

5) *Efficiency_Test*: compare the size of comresed file with the original document. Figure 14 (a) repesents the size of compressed file and Figure 14 (b) the final result.

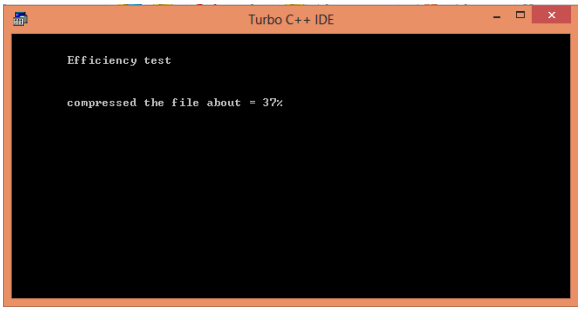

Fig. 14(a). Processing of document(Equality_Test)

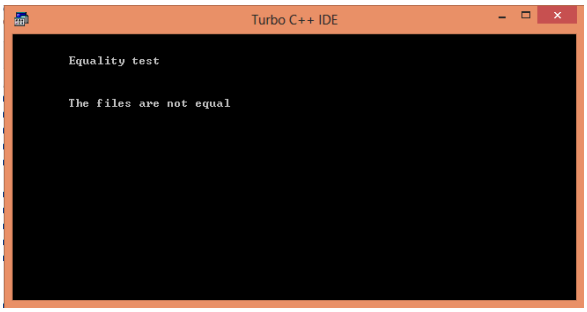

Fig. 14(b). Processing of document(Equality_Test Result)

```
cout<<"\n\n\tEquality test\n\n";
 char ch1,ch2;
 ifile2.open("decomp.txt",ios::in);
 ifile1.open(f1,ios::in);
 ifile1.get(ch1);
 ifile2.get(ch2);
 while(ifile1 && ifile2 && ch1==ch2)
  {
   ifile1.get(ch1);
   ifile2.get(ch2);  //cout<<ch1<<ch2<<"\n";
  }

  if(ifile1==ifile2)
  cout<<"\n\n\tThe files are equal";
  else
  cout<<"\n\n\tThe files are not equal";

  ifile1.close();
  ifile2.close();
```

Fig. 15. Implementation Code for equality test

Figure 15 depicts the implementation code for original file and compressed file size comparison that will compare the size of both files and will result either both files are equal or not.

From the above results it is clear that the document is compressed 37%. Such method can be used to utilize the repository.

VI. Conclusion

The proposed work presentes Clustered Webbase search engine architecture, supporting clustered web repository, and meta-data management. Clustered Webbase search engine architecture uses repository and working modules to distribute data among domain specific clusters which compose a clustered web repository having the following characteristics-

(1) **Full Distribution:** Distribution of web pages to different domain specific blocks reduces searching complexity.
(2) **Capacity improvement:** The proposed work also representing an indexing mechanism to store the keywords present in the document in compressed form by mapping variable length Huffman code. It also focuses on the presence of keyword in different document because the keyword in maximum number of documents will be mapped to less length Huffman code thus mechanism reduces the size of document and after updates the index.
(3) **Fast search:** The data structure of the indexer fastens the search for matched results from the Inverted Index with the cluster information. It also helps the user to process the user query with fast and more relevant results.

Future work on Clustered Webbase search engine includes a detailed study on performance issues, working complexity of different modules and management improvement of extensible metadata.

REFRENCES

[1] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", International Conference on Computer networks and ISDN systems, pp. 107-177,1998.
[2] Ashutosh Dixit and Niraj Singhal," Web Crawling Techniques : A Review", National Seminar on Information Security: Emerging Threats and Innovations in 21st Century, pp. 34-43, 2009.
[3] Nidhi Tyagi and Deepti Gupta, "A Novel Architecture for Domain Specific Parallel Crawler", Indian Journal of Computer Science and Engineering, Vol 1 No 1 44-53,2010.
[4] Jun Hirai Sriram Raghavan Hector Garcia-Molina Andreas Paepcke,"WebBase : A repository of web pages", Ninth International World Web Conference , pp. 277-293 ,2000.
[5] Gray, M., "Internet Statistics: Growth and Usage of the Web and the Internet", http://www.mit.edu/people/mkgray/net/,1996.
[6] Burner, M., Crawling towards Eternity: "Building An Archive of The World Wide Web", Web Techniques Magazine, Vol. 2, No. 5, 37-40,1997.
[7] Shikha Maan and Mukesh Rawat "Design and Implementation of Specialized form Focused Crawler" , International Journal of Contemporary Research in Engineering & Technology, Vol. 3, No. 1&2,2013
[8] Kavita Saini "Extraction of Lables from Search Interfaces for Domain Specific HiWE", International Journal of Contemporary Research in Engineering & Technology, Vol. 3, No. 1&2,2013
[9] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina and Andreas Paepcke "WebBase : A repository of web pages", Ninth International World Web Conference ,pp. 277-293 ,2000.
[10] Jinru He, Hao Yan, Torsten Suel, "Compact Full-Text Indexing of Versioned Document Collections ", ACM conference on Information and knowledge management,pp. 415-424, 2009.
[11] Khaled M. Hammouda and Mohamed S. Kamel, "Efficient Phrase-Based Document Indexing for Web Document Clustering" , IEEE transactions on knowledge and data engineering, vol. 16, no. 10, pp. 1279-1296 , 2004.
[12] Pooja Mudgil, A. K. Sharma and Pooja Gupta "An Improved Indexing Mechanism to Index Web Documents", International Conference on Computational Intelligence and Communication Networks, pp. 460-464, 2013.
[13] Geeta Rani and Nidhi Tyagi "A Cluster Balancing Approach for Efficient Storage of Web Documents" National conference on recent trends in advanced computing, electronics and information technology, CICON-2014.
[14] Cho, J., Garcia-Molina, H., Haveliwala, T., Lam, W.,Paepcke, A., Raghavan, S., Wesley, G., "WebBase Components and Applications",ACM Transactions on Internet Technology,pp. 153-186 ,2006.
[15] Frank McCown and Michael L. Nelson, "Evaluation of Crawling Policies for a Web -Repository Crawler", 17th conference on Hypertext and hypermedia, pp. 157-168,2006,
[16] Konstantin Shvachko, Hairong Kuang, Sanjay Radia and Robert Chansler, "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies , pp. 1-10, 2010.
[17] Jeffrey Dean and Sanjay Ghemawat, " MapReduce: Simplied Data Processing on Large Clusters", *OSDI* , Vol 51,pp. 107-113 , 2004.
[18] Junghoo Cho and Hector Garcia-Molina" The Evolution of the Web and Implications for an Incremental Crawler", International Conference on very large database, pp. 200-209, 2000.
[19] Burner, M., "Crawling towards Eternity: Building An Archive of The World Wide Web", Web Techniques Magazine, Vol. 2, No. 5, pp. 37-40,1997.
[20] http://en.wikipedia.org/wiki/Domain-name
[21] Fred Douglis, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios, *WebGUIDE:* Querying and Navigating Changes in Web Repositories, International Conference world wide web,pp. 6-10 , 1996.
[22] Lorna M. Campbell, Kerry Blinco and Jon Mason, "Repository Management and Implementation", Altilab04-repositories ,11 July 2004.

Authors' Profiles

Geeta Rani has completed her B.Tech, computer science degree from Uttar Pradesh Technical University in 2012 and received her M.Tech. in Computer Engineering in july 2014 from Shobhit University. Presently. Presently, she is working as Assistant Professor in Department of Computer Science and Engineering at Shobhit University, Meerut. Her area of interest includes Computer Network, Database Management System and Information Retrieval.

Nidhi Tyagi, received her M.Tech. in Computer Engineering (first class with Honours) in 2009, and Ph.D. in Computer Engineering in 2013, from Shobhit University, Meerut. Presently, she is working as Associate Professor in Department of Computer Science and Engineering at Shobhit University, Meerut. She has a teaching experience of more than 13 years. Her research interest includes Search Engine, Crawlers and Data Mining.