An Efficient Multi-Threaded Web Crawler Using HashMaps

Yasin Kansu, Begum Mutlu, Anıl Utku, and M. Ali Akcayol

Abstract—**In last decades, the number of web pages on the Internet has been exposed a rapid increase intrinsically, and the information on the Internet has reached a very large size. Search engines have been developed to access this large-scale information efficiently. Web crawlers play a very important role in search engines. In this paper, an efficient multi-threaded web crawler is proposed, and empirically analyzed in terms of crawling speed and coverage.**

Index Terms—**Coverage, HashMap, web crawler.**

I. Introduction

A web search engine has been used to search for information obtained from text, images, videos etc. on the World Wide Web. The search engines use crawlers to gather and update its database include indices and contents of web pages.

Searching on the web has a great challenge due to the rapid growth of information. [1]. In the literature, the problem has been divided into two parts as crawling and indexing of the web pages in a very short time and for a wide range of coverage.

The first web crawler has been presented in 1993 by Matthew Gray [1]. Numerous successful methods have been developed from that day. In recent years, some discussions have been raised about how often the scanned web pages need to be re-scanned due to effecting on crawling performance [2]. Boldi *et al.* have proposed an efficient solution to improve performance of crawling process using distributed multiple crawlers running in parallel [3], [4]. By optimizing the number of parallel agents, performance of the crawler can be remarkably improved [5], [6].

In [7], an artificial intelligence method has been proposed to improve performance of crawling process. In this work, the proposed crawler called MySpiders scans web pages using multi-threaded structure. Similar architecture was used in [8] as well.

Today, many researchers have studied on crawling mechanism [9], refresh policies [10], visit strategy for newborn links [11]. In [10], Internet traffic has been investigated caused by refresh policies of web crawlers. The authors have investigated on how they can maintain local copies of original data sources up to date. They proposed a new refresh policy and examined its effectiveness experimentally.

In [9], the traffic caused by web crawlers of the major search engines was examined. More recently, a web crawler named IWatch has been designed to analyze the distribution of information on the Internet [12].

In this paper, an efficient multi-threaded web crawler algorithm has been developed using HashMaps. The main steps of the study are accessing the web pages, saving their contents, creating a directory, and performing an efficient search. The inverted index has been used to keep local summarized copy of original web pages. The web pages have been crawled and newborn web pages have been detected periodically.

II. Implementation

In this section, design details of proposed web crawler algorithm are presented with regards to its strategy on crawling web pages, keeping summarized contents of these pages and searching for items on the web pages.

A. Multi-Threaded Crawler

The multi-threaded crawler initially visits the web pages by starting from the seed fields. The URLs in these pages are extracted and inserted into the queue. Thus, all URLs can be accessed in the web pages. Fig. 1 shows structure of the multi-threaded crawler.

As shown in Fig. 1, the multi-threaded crawler consists of (i) a component that manages shared areas, (ii) a component that scans pages, and (iii) a control component that organizes these components. It inserts the web addresses into the queue of manager component. It also adds the hosts of the start addresses to the host routing table of the manager and initializes the specified number of crawler components.

The crawlers visit the web pages supplied by the link pool manager. New page addresses determined from visited links are delivered to the manager. Manager checks the links whether they were visited before or not, and inserts them into the queue if it is necessary repeatedly until the process is finished by the user. When any crawler needs to get an address, link pool manager gets a link from the queue and adds to the visited links. Simultaneously, the host of the link is added to the list of hosts. If the host has already visited before, it only increases the number of visits for the corresponding host.

In order to reach different hosts, the URLs are divided into two parts as interlinks and intra-links. The intra-links point to the same host, otherwise the interlinks point to the other hosts. All the links are inserted to a priority queue that have higher priority for interlinks than intra-links.

Regarding the aforementioned lists of hosts, a red-black tree data structure is utilized to manage the list of hosts, list of

Manuscript received August 14, 2017; revised October 20, 15.

Y. Kansu is with the Command Control and Combat Systems, Havelsan, Ankara, Turkey (e-mail: ykansu@ havelsan.com.tr).

B. Mutlu, A. Utku and M. Ali Akcayol are with the Department of Computer Engineering, Gazi University, Ankara, 06570 Turkey (e-mail: begummutlu@gazi.edu.tr, anilutku@gazi.edu.tr, akcayol@gazi.edu.tr).

doi: 10.18178/jacn.2017.5.2.242

visited links and list of standby hosts. The reason behind this implementation is that it has low computational cost for searching, insertion and deletion tasks of items. Otherwise, crawlers in a multi-threaded structure could have to wait for other processes.

The link pool manager inserts the links of the hosts to the priority queue or may redirect the link if required. The counter of the links belongs to visited host is incremented and the value field in red-black tree is updated. Since this process is performed in the host table, number of visited links of a host is automatically computed.

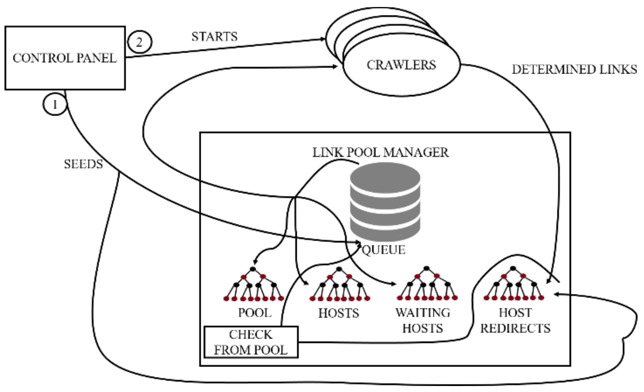

Fig. 1. Proposed multi-thread crawler architecture.

B. Indexer

The web pages that are visited by the multi-threaded crawler are all sent to the indexer. Fig. 2 shows the pseudocode the main tasks of the developed multithreaded crawler.

The contents of the pages are recorded, once their addresses are inserted into priority queue (in line 5). The indexer extracts head and body sections of the web pages (in line 6) which are then stored in a specially constructed HashMap data structure.

The HashMap data structure allows storing data, which consists of key-value relationship and mapping key to value at $O(1)$ time complexity. Therefore, computational cost of the HashMap structure is very low for searching, adding, deleting and editing. It should be noted that in order to achieve high speed for access, mapping key-value has to be unique.

The extracted words are used as keys. A new HashMap structure is created to store value. This HashMap holds the page address as the key and a private class which keeps some data about the relationship between the word and the web page address as the value. The HashMap structure is expressed by (1).

$$HashMap\{k|HashMap\{url|(c,n,TF)\}\} \tag{1}$$

In expression (1), *k* is the word, *url* is the web page address, *c* is the first sentence of the word on the page, *n* is the number of occurrences of the word on the page. Additionally, *TF* stands for term frequency which represents the ratio of the number of occurrences of the word on the page to the total number of words on the page. The process for a web page begins with the parsing web page and extract words. The terms are checked in the HashMap. If a term exists, the value of the term is taken, otherwise a new value for HashMap is created and the word is added to table. For existing page, the process is finished by increasing the number of occurrences by 1 and updating the *TF* value (in line 14). Otherwise, a sentence is obtained with words before and after 10th place on the page where the word is passed, and this sentence is appended to the value of the word HashMap with the number of passes of the word 1 and the calculated TF value as well as the address of the value page (in line 21 in Fig. 2). All search operations are performed by the HashMap at $O(1)$ time complexities that no matter with how many records are there. The developed data structure can be seen as growing list down as new words are inserted. If users are not interested in the page addresses ranking by search engines, the list can grow right to avoid unnecessary use of system resources and allocate more resources to grow down. For this reason, growth to the right in the data structure is limited to 100 pages (in line 18).

It is necessary to compare the address to be added with the existing addresses in the list in case of bounded words (in line 19). Since the *TF* cannot be calculated correctly when a new address is added for the first time, the page with the smallest *TF* is extracted directly from the pages and the new page is added.

```
 1:                                                              ▷ CRAWLER OBJECT
 2: STORE(Page content as HTML)
 3:                                                              ▷ MANAGER OBJECT
 4: procedure STORE(content)
 5:     ADDRETRIEVEDLINKSTOQUEUE(content.links)
 6:     head = content.head
 7:     body = content. body
 8:     for all word ∈ head and body do
 9:         ADDDATASTRUCTURE(word)
10:     end for
11: end procedure
12: procedure ADDDATASTRUCTURE(word)
13:     if structure contains word then
14:         if link had been attached the word before then
15:             increase the number of occurances
16:             update $TF_{word}$
17:         end if
18:         if word had been added more than 100 times then
19:             remove the link attached attached to this word with least $TF_{word}$
20:         end if
21:         append link to word with extra information about link
22:     end if
23:     if structure does not contain word then
24:         add word as new substructure
25:         append link to word with extra information about link
26:     end if
27: end procedure
28: procedure ADDRETRIEVEDLINKSTOQUEUE(Sublinks)
29:     for all hosts ∈ Sublinks do
30:         if host had been already added to redirection table then
31:             increase redirection number of host
32:         end if
33:         if host had not been already added to redirection table then
34:             add host to redirection table with value 1
35:         end if
36:     end for
37:     for all sublink ∈ Sublinks do
38:         if sublink is not visited then
39:             enqueue (Sublink)
40:         end if
41:     end for
42: end procedure
```

Fig. 2. A representation of main tasks of crawler as pseudocode.

C. Searchers

The searcher is designed to list the addresses of the web pages containing the desired search words given by the user. Some information has been created and recorded in the other components so that the list displayed to the user can be sorted.

The searcher also performs a ranking by using this information.

The searcher component starts processing by separating the user-entered text words and getting results for all words. Using HashMap data structure, this process is completed without any delay. Then, for each word, HashMaps contain up to 100 page links per word (for the most extreme case with 100 records per word). If there are common links between these HashMaps, these links contain multiple words entered by the user at the same time. The searcher checks the hierarchy in HashMaps to find these common page addresses. Since the HashMap searches at time $O(1)$ and the number of the HashMap registers is 100, all the elements of one HashMap are checked at $100 \times O(1)$ times in the other HashMaps. In the worst case, this means a check up to 100 times the number of words.

For this reason, when more than one word is searched, the control time near $O(1)$ speed $(200 \times O(1), 300 \times O(1))$ is calculated. This is independent of the size of the index, depending on the number of words desired to be searched. In addition, since links containing more than one word are checked once, they don't need to be checked again. So, a representing the number of words searched, v representing the number of check operation, the number of check operation is represented by (2).

$$O(1) \times a \times 100 > v \qquad (2)$$

As the number of links containing more than one word increases, the speed of control decreases.

As a result of the check operation, a new TreeMap data structure holds the number that how many common words link is related as keys and values in a HashMap that contains link and other data. The TreeMap structure is expressed by (3).

$$TreeMap\{o \,|\, HashMap\{url\,|\,(c, n, TF)\}\} \qquad (3)$$

In equation (3) o represents the number of searched words links are related. Since the TreeMap data structure automatically ordered by keys, it does not need to be sort again. If the tree was traversed bottom to up, the links containing the most searched words are first obtained.

In the second stage, the links in the HashMaps of the TreeMap data structure are sorted by processing HashMaps by creating a sort number with various parameters. Using this number, the HashMaps are converted to a simple TreeMap. The structure of the TreeMap is given in (4).

$$TreeMap\{s \,|\, \{url, c\}\}. \qquad (4)$$

In equation (4), url is the address of the page, c sentence include the word, s is expressed to represent the number for sorts.

Finally, the links are completely sorted. The parameters are the TF value, the number of times the linking host address is routed to different host addresses, the number of lower links which are possessed by host address, the number of links at the link address, and the number of links at the host address. The parameters have an effect on ordering determined by their coefficients. Sorting can be controlled by changing the coefficients and the most suitable coefficient values can be determined.

III. Experiments & Results

The experiments are performed on Intel Core i3 3217U 1.80 GHz processor running at 8 GB RAM at 1333 MHz. The storage equipment is SATA3 connected local storage device with 5400 rpm. In addition, Internet connection has 3Mbps for download and 5Mbps for upload. It has been observed that threads usually spend most of the processing time to retrieve data on the Internet.

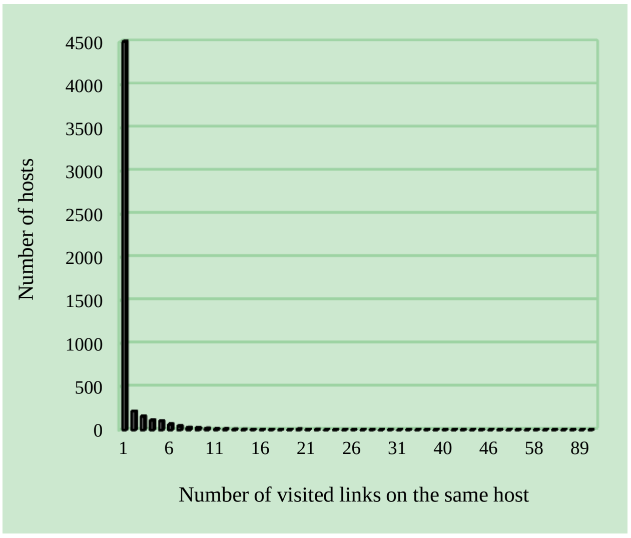

Fig. 3. Relation between the number of hosts and the number of visited links per each host.

The first analysis is related with how many links are visited by the hosts on the host list. The change in the number of visited links on the same host according to the number of hosts is shown in Fig. 3. As can be seen in here, the number of pages visited by a single link is very high. This shows that, aimed selectiveness about retrieving a new link form the queue is achieved, interlink priority queue succeeded.

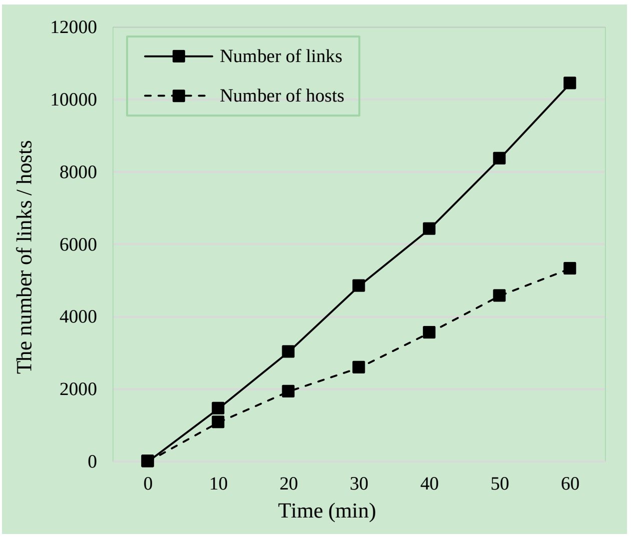

Fig. 4. An illustration of the increase in the number of hosts and links by time.

The number of hosts and the number of web pages visited in 10 minutes is presented in Fig. 4. As can be seen in here, the number of visited hosts and web pages is increasing linearly and the gap between them is proportionally same. If the proportional difference between the number of hosts and the number of links increases in time, it indicates that mostly the links on the same host are visited.

The change number of the links and hosts according to number of parallel crawler is shown in Fig. 5. It can be seen that the excessive number of crawlers can cause adverse effects on the application performance. More than 100 parallel crawlers have increased negative impact and performance has begun to fall. It is shown in Fig. 5 that using 8 crawlers is also suitable for Internet bandwidth. When 8 crawlers are used, the maximum number of hosts is not reached.

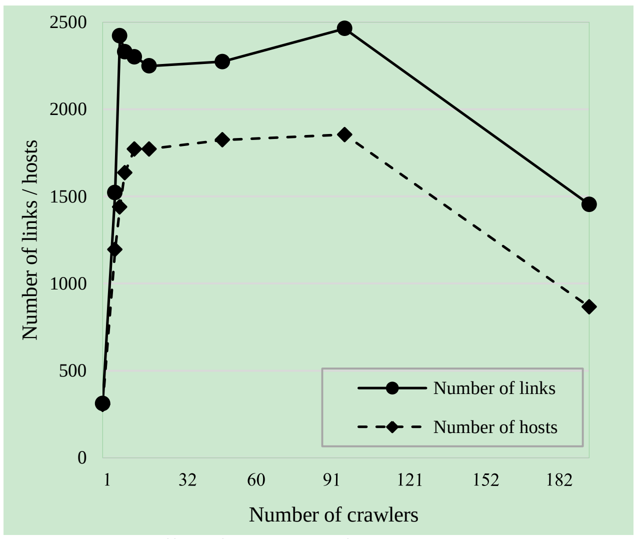

Fig. 5. The effect of the number of crawlers on the visited links.

TABLE I: HOSTS WITH MAXIMUM NUMBER OF REDIRECTIONS FROM DIFFERENT HOSTS

Host	Number of redirection
www.about.com	31.808
twitter.com	20.933
www.yourdictionary.com	19.601
www.facebook.com	13.028
wikimediafoundation.org	8.401
www.blogger.com	8.064
en.wikipedia.org	8.001
www.stateuniversity.com	7.682
www.mediawiki.org	6.821
incubator.wikimedia.org	6.453

Regarding the number of visits, the hub pages are given in Table I. Here, the most linked pages are social media and encyclopedia pages. In this analysis, *www.about.com* takes the lead because it has its own sub-links on different hosts, such as *breakfast.about.com*, *gardening.about.com*. It may also be directed from different host addresses, but the sub-links are the majority. Results of *Twitter* and *Facebook* are plausible and foreseeable, since almost every web site keeps *Facebook* and *Twitter*. The encyclopedic sites also have high results due to the some of starting addresses have encyclopedic sites. The results show that the crawled pages are gathered from very large area.

In the last analysis, the number of recorded page, the number of recorded word and the speed of searching were obtained within 10 minutes for query text "news Turkey". The results are shown in Table II. According to results, search length in the directory is independent of the size of the directory.

TABLE II: 10-MINUTE SEARCH PERFORMANCE

Number of pages on record	Number of words on record	Search speed in record (ms)
1.345	116.367	3,9156
2.100	126.425	4,5022
2.814	135.767	3,4405
3.103	143.726	3,9833
3.851	155.220	3,0180
4.295	161.449	4,1629

IV. CONCLUSION

Due to the rapid increase in the number of web pages, it is inefficient and implausible to process all of these pages, and gathering information about their contents in order to search the web pages. It is not possible to be fully informed of web pages because the servers that provide access to the web pages are distributed. Also, it is not possible to know the addresses and contents of all pages. For this reason, scanning and saving these pages with web crawler software is important to increase the speed of accessing useful information.

In this study, an efficient multi-threaded web crawler architecture is proposed. The web pages are obtained by the crawler algorithm, and stored in red-black tree structure for efficient insertion, deletion and searching performance. The HashMap data structure has been used for key-value mapping. Various analyses are performed by using the proposed web crawler software in terms of crawling speed and coverage; and results are examined with deep discussions.

REFERENCES

[1] A. Heydon and M. Najork, "No title," *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.

[2] J. Edwards, K. McCurley, and J. Tomlin, "An adaptive model for optimizing performance of an incremental web crawler," in *Proc. the Tenth International Conference on World Wide Web - WWW '01*, 2011, pp. 106–113.

[3] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "UbiCrawler: A scalable fully distributed web crawler," *Softw. - Pract. Exp.*, vol. 34, no. 8, pp. 711–726, 2004.

[4] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Trovatore: Towards a highly scalable distributed web crawler," *WWW Posters*, pp. 7–8, 2001.

[5] V. Shkapenyuk and T. Suel, "Design and implementation of a high-performance distributed Web crawler," *Icde 2002*, pp. 357–368, 2002.

[6] D. Chau, S. Pandit, S. Wang, and C. Faloutsos, "Parallel crawling for online social networks," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 1283–1284.

[7] G. Pant and F. Menczer, "MySpiders: Evolve your own intelligent web crawlers," *Auton. Agent. Multi. Agent. Syst.*, vol. 5, no. 2, pp. 221–229, 2002.

[8] A. Rungsawang and N. Angkawattanawit, "Learnable topic-specific web crawler," *J. Netw. Comput. Appl.*, vol. 28, no. 2, pp. 97–114, 2005.

[9] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou, "An investigation of web crawler behavior: Characterization and metrics," *Comput. Commun.*, vol. 28, no. 8, pp. 880–897, 2005.

[10] J. Cho and H. Garcia-Molina, "Effective page refresh policies for web crawlers," *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 390–426, 2003.

[11] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez, "Crawling a country: Better strategies than breadth-first for web page ordering," *Spec. Interes. Tracks Posters 14th Int. Conf. World Wide Web*, pp. 864–872, 2005.

[12] C. Jensen, C. Sarkar, C. Jensen, and C. Potts, "Tracking website data-collection and privacy practices with the iWatch web crawler," in *Proc. Third Symp. Usable Priv. Secur.*, 2007, pp. 29–40.

Yasin Kansu received the B.S degree from Gazi University, Department of Computer Engineering in 2016. He is currently software engineer in command control and combat system, Havelsan, Ankara, Turkey. His research interests include recommendation systems, big data analysis, web search engines, web crawlers.

Begum Mutlu is the corresponding author, she received her B.S degree from Gazi University, Department of Computer Engineering in 2012. She received M.Sc degree in Computer Engineering Department, Hacettepe University in 2014. She is currently a Ph.D candidate and research assistant at Department of Computer Engineering, Gazi University. Her research interests are soft computing, deep learning, and text mining.

M. Ali Akcayol received the B.S degree in electronics and computer systems education from Gazi University in 1993. He received M.Sc and Ph.D degrees in Institute of Science and Technology from Gazi University in 1998 and 2001, respectively. His research interests include mobile wireless networking, web technologies, web mining, cloud computing, artificial intelligence, intelligent optimization techniques, hybrid intelligent systems.

Anıl Utku received the B.S degree from Kocaeli University, Department of Computer Engineering in 2010. He received M.c degree from Computer Engineering Department, Gazi University in 2015. He is currently a Ph.D candidate and research assistant at Department of Computer Engineering, Gazi University. His research interests include recommendation systems, big data analysis and wireless sensor networks.