Efficient watcher based web crawler design

Saed Alqaraleh, Omar Ramadan and Muhammed Salamah
Eastern Mediterranean University, Famagusta, Turkey

Received 10 February 2015
Revised 11 September 2015
Accepted 23 September 2015

Abstract

Purpose – The purpose of this paper is to design a watcher-based crawler (WBC) that has the ability of crawling static and dynamic web sites, and can download only the updated and newly added web pages.

Design/methodology/approach – In the proposed WBC crawler, a watcher file, which can be uploaded to the web sites servers, prepares a report that contains the addresses of the updated and the newly added web pages. In addition, the WBC is split into five units, where each unit is responsible for performing a specific crawling process.

Findings – Several experiments have been conducted and it has been observed that the proposed WBC increases the number of uniquely visited static and dynamic web sites as compared with the existing crawling techniques. In addition, the proposed watcher file not only allows the crawlers to visit the updated and newly web pages, but also solves the crawlers overlapping and communication problems.

Originality/value – The proposed WBC performs all crawling processes in the sense that it detects all updated and newly added pages automatically without any human explicit intervention or downloading the entire web sites.

Keywords Information retrieval, Search engine, AJAX crawler, Crawler re-visiting policies, Crawling algorithm, Static crawler

Paper type Research paper

1. Introduction

Nowadays, the search engine crawlers, which are used to collect information in a routinely manner, have been considered to be one of the most important search engine components. It is an automatic web object retrieval system that exploits the web's link structure. It has two primary goals: first, finding new web objects; and second, observing changes in previously discovered web objects (Agarwal *et al.*, 2012; Uzun *et al.*, 2013; Amolochitis *et al.*, 2013). To achieve the first goal, the crawler has to visit as many web sites as possible, and to achieve the second goal, the crawler has to maintain the freshness of the previously visited web sites, which can be achieved by re-visiting such web sites in a routinely manner. In the following, the most frequently used re-visiting policies are summarized:

(1) *Uniform policy*: in this policy, the entire web sites are downloaded at each visit (Bhute and Meshram, 2010; Pichler *et al.*, 2011; Leng *et al.*, 2011; Sharma *et al.*, 2012; Singh and Vikasn, 2014). Although this approach enriches the databases, it requires a large processing time.

(2) *Proportional policy*: this policy is performed in many ways, such as:

- Downloading only the pages that have a rank more than a threshold value specified by the crawler administrator (Bhute and Meshram, 2010;

The authors would like to thank Assistant Professor Dr Yıltan Bitirim (Eastern Mediterranean University) for his valuable suggestions and comments that greatly improved the manuscript.

Aslib Journal of Information Management
Vol. 67 No. 6, 2015
pp. 663-686

2050-3806
DOI 10.1108/AJIM-02-2015-0019

Pichler *et al.*, 2011; Leng *et al.*, 2011). The rank of a page is based on the importance and the frequency of updating that page.

- Downloading the web pages allocated in the top N levels of each web site (Pichler *et al.*, 2011; Cho *et al.*, 2012). In general, this type of proportional policy which is based on breadth first algorithm, involves visiting the main page (root) of the web site, and downloading only the pages that have URL links or allocated in the top N levels. This helps the crawler to avoid exploring too deeply into any visited web site (Olston and Najork, 2010).

It is important to note that the time required for re-visiting a web site for the proportional policy is significantly less than the uniform policy. On the other hand, the proportional policy may ignore visiting the new web pages, as their rank is initially low, and it may also ignore the updated web pages which are not allocated on top N levels.

In recent years, rich internet applications (RIAs) (Bruno, 2006), that enhance and support the accessibility of scripted and dynamic contents, become more and more popular. AJAX, which is a group of interrelated techniques that can be used on the client side to create asynchronous web applications, can be considered as the most popular technique used in RIAs (Bruno, 2006).

From the above survey, it can be concluded that the web pages can be categorized into the following two categorizes: static and dynamic web pages. In the static case, web pages are allocated in the web site's server and delivered to the user exactly as stored on the server web site (Cui *et al.*, 2013). In the dynamic case, the web pages use the AJAX techniques. In addition, triggering AJAX events may dynamically introduce new pages, known as states, and in most cases these pages are not allocated on the server. It should be noted that the process of updating static pages can be done by editing and changing the contents of the file offline. On the other hand, the content of AJAX pages can be generated and updated dynamically (online) without reloading the whole page or changing the URL.

1.1 Crawlers challenging problems

In this section, the crawler challenging problems are summarized. First, most of current crawler techniques cannot detect the updated pages online (Agarwal *et al.*, 2012; Uzun *et al.*, 2013; Amolochitis *et al.*, 2013), and this will require the crawler to download all the web pages to detect the updated ones. Second, overlapping problem occur when more than one crawler process the same web page. Third, up to our knowledge, most of the crawling techniques require communication between the running crawlers which increases the crawling processing time and requires high-quality networks (Mukhopadhyay *et al.*, 2006; Wu and Lai, 2010; Kumar and Neelima, 2011; Agarwal *et al.*, 2012; Amolochitis *et al.*, 2013; Uzun *et al.*, 2013). The fourth crawling problem is that the conventional crawlers work is based on the URLs, and download only the pages that are allocated on the web site server, and therefore, they are inefficient when dealing with AJAX pages, as they cannot index the web sites dynamic information (Mishra *et al.*, 2010; Nath and Bal, 2011; Bhushan *et al.*, 2012). In addition to the above static crawling problems, AJAX crawling techniques are still suffering from several challenging problems such as the following: first, identifying web page's states – in some cases, in order to identify the page states the AJAX events need to be triggered, and this may lead to change the content of the corresponding page without changing the page URL, and such page will be recognized as one of the page's states. Second, as AJAX technique contains many

events, triggering all page events will lead to a very large number of states and in some cases, more than one event may lead to the same state, and this will be considered as duplicated states. For example, by clicking the "next" tab in page ($i-1$) and clicking the "previous" tab in page ($i+1$), both will lead to the same page (i), i.e., the same state (Duda *et al.*, 2009).

1.2 Related work

In the last decade, several developments in the web crawler field have been introduced. Mukhopadhyay *et al.* (2006) introduced a dynamic parallel web crawler based on client-server model, named as WEB-SAILOR. This approach eliminates the communication between the running crawlers by introducing a seed-server which is responsible for the crawling decisions. Wu and Lai (2010) studied a crawler based on the cluster environment. In this approach, a new distributed controller pattern and dynamic assignment structure were used. Kumar and Neelima (2011) implemented a new crawler, named as DCrawler. In this crawler a new assignment function is used for partitioning the domain between the crawlers. Amin *et al.* (2012) proposed a scalable web crawling system, named as WEBTracker, to increase the number of visited pages by making use of distributed environment. Topic specific crawlers, also known as focussed crawlers have been developed in Mukhopadhyay and Sinha (2008) and Yang *et al.* (2009). This type of crawlers downloads only web pages that are relevant to one or more pre-defined topics.

Although the above approaches have improved the performance of web crawlers, the number of web sites that can be processed by the crawlers is still limited compared with the current huge number of published web sites. In addition, the conventional crawler techniques require downloading all web site pages to find the updated ones, and this will increase the internet traffic and the bandwidth consumption. It has been found that approximately 40 percent of the current internet traffic, bandwidth consumption and web requests are due to search engine crawlers (Mishra *et al.*, 2010; Nath and Bal, 2011). To solve this issue, mobile crawlers (Mishra *et al.*, 2010; Nath and Bal, 2011) and sitemaps-based crawlers (Schonfeld and Shivakumar, 2009; Bhushan *et al.*, 2012; Brawer *et al.*, 2013) were introduced. The details of these two techniques are explained below:

(1) *Mobile crawlers*: unlike the conventional crawlers, mobile crawlers go through the servers of the URLs in its frontier to detect and store in its memory the required data such as the updated pages. This mechanism reduces the amount of data transferred over the network and therefore, decrease the network load caused by the crawlers. However, mobile crawling techniques are not widely used due to the following problems: first, mobile crawlers occupy large portions of the visited web site resources such as memory, the network bandwidth and CPU cycles, etc. (Nath and Bal, 2011). Second, due to security reasons, the remote system in most cases will not allow the mobile crawlers to reside in its memory, and may recognize it as viruses.

(2) *Sitemaps-based crawlers*: a sitemap file is an XML file that contains a list of the URLs of web site pages with additional metadata such as: *loc* which is a required field representing the URL of the web page; and *lastmod*, which is an optional field representing the last time the web page of the URL was updated. Recently, the web sites start providing sitemap(s) to the users for easier navigation. Sitemaps-based crawlers use the sitemap(s) to find the updated and the new

web pages. However, this mechanism suffers from the following problems: first, many web sites do not have sitemap, and therefore, the web crawler has to follow the traditional way of crawling; second, recently, a group of web sites and software have been developed to create the web site's sitemaps automatically. However, when any of the site's pages is updated, the system administrator has to update the page's metadata such as *lastmod* manually (Mishra *et al.*, 2010; Bhushan *et al.*, 2012). This makes the process of updating the sitemap, especially for larger web sites, difficult and time consuming.

In addition to static web pages indexing crawlers, dynamic indexing crawlers were also introduced. In Sharma *et al.* (2010), a new crawler, which extracts the information in dynamic pages by analyzing javascript language, was introduced. In Yao *et al.* (2012), an ontology-based web crawler that can download the information in dynamic pages has been proposed. In Cui *et al.* (2013), an AJAX crawler that crawls dynamic web pages has been developed. In Dincturk *et al.* (2014), a new crawling methodology, named as model-based crawling, was introduced to design efficient crawling strategies for RIAs. Although current AJAX-based crawlers can extract a promising percentage of dynamic pages information, it is important to note that this process still requires a large processing time and this will slow down the process of extracting dynamic data (Sharma *et al.*, 2010; Yao *et al.*, 2012; Cui *et al.*, 2013; Dincturk *et al.*, 2014).

In addition to the above crawling techniques, it is important to note that most of the information about the commercial search engines crawlers such as Google and Bing are kept hidden as business secrets, and there are very few documents about the mechanisms of these crawlers.

The remainder of this paper is organized as follows: Section 2 presents the developed watcher-based crawler (WBC) approach. The discussion of the experimental study and results are presented in Section 3, and finally, conclusions are given in Section 4.

2. WBC structure

The developed WBC consists of two main parts: the watcher file; and the WBC server. The structure of these parts is shown in Figure 1, and their details are summarized below.

2.1 The watcher file

In the proposed WBC, the watcher file, which will be uploaded to the web sites servers, prepares a report that contains only the updated and the newly added web pages. The watcher file is small in size which does not require any specific requirement on the web server and it will not affect its performance. The main advantage of the watcher file is that it allows the crawlers to visit only the updated and the newly added pages. In addition, it allows solving crawlers overlapping and communication problems by introducing a flag in the watcher report, which will be set to 1 by the watcher file when a crawler processes that web site. In this case, other copies of WBC will not visit any web site whose flag is set. Hence, there will be no need for communication between the running crawlers.

2.1.1 Mechanism of building the watcher's report. The mechanism of building the watcher report is performed using the following monitoring and ranking functions:

(1) *Monitoring function*: this function keeps track of the web site directories and detects the updated and the newly added pages. In the case of updating static

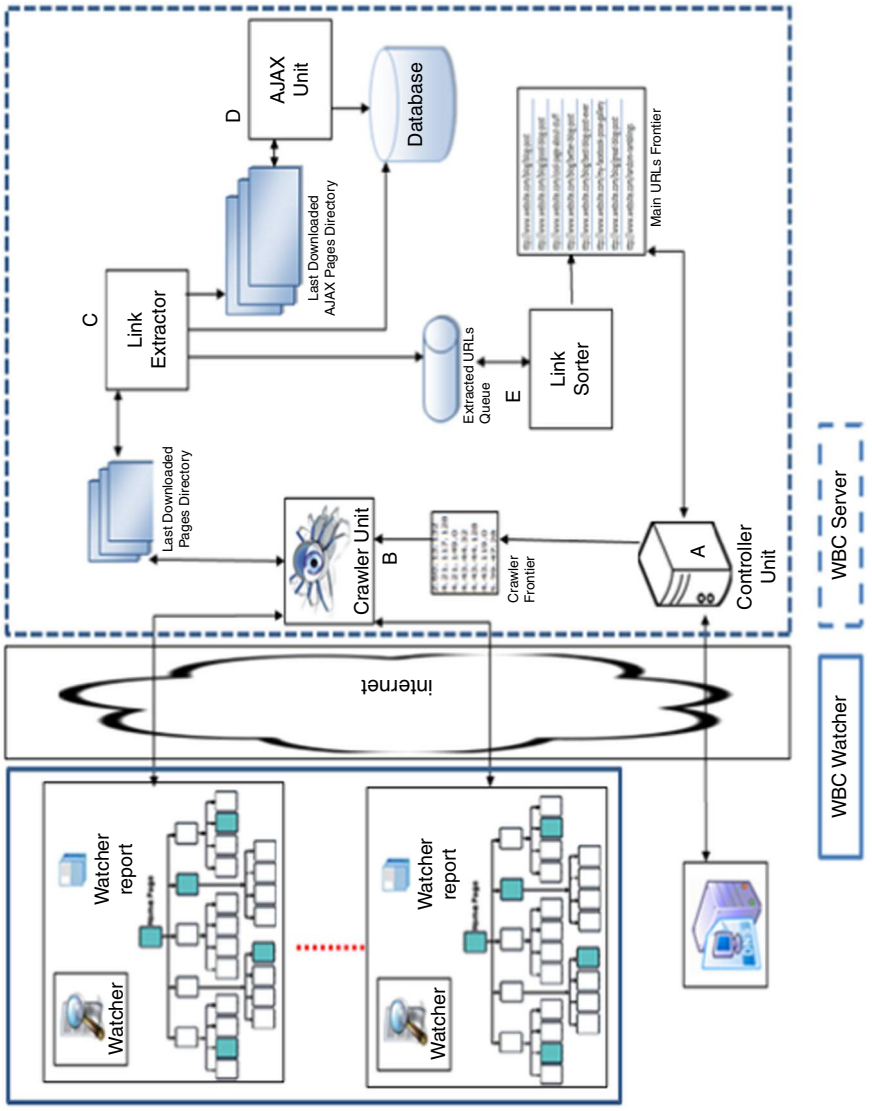


Figure 1.
WBC structure

pages, the ranking function will be called to rank and add the pages URLs in the appropriate position in the report. In the case of dynamic pages, the monitoring function detects and adds the AJAX events including its triggering path into the watcher report as shown in Figure 2. The format of adding the dynamic page information is:

$$\textit{Dynamic page URL},\ \textit{Event}(s),\ \underbrace{\textit{Source element}(s),\ \textit{Target element},\ \textit{Value}}_{\textit{Triggering path}}$$

where *Event*(s) is the javascript event that will be triggered, *Source element(s)* represents the corresponding HTML object, *Target element* is the object whose information will be updated, and *Value* is the new value that will be assigned to the target or the name of the function that will be called. For instance, a dynamic page event can be represented in the report as follows:

www.test.com/main.asp,	**onclick**,	**div id = "next"**,	**text.innerHTML**,	**Updateinfo()**
↑	↑	↑	↑	↑
Dynamic page URL	*Event*	*Source*	*Target*	*Value*

As mentioned before, one AJAX crawling problems is that the same state may be retrieved multiple times. The following two scenarios illustrate this problem: first, in some cases, more than one HTML object in an AJAX page may contain exactly the same event and its triggering path. For instance, by triggering the following two reported events, the same state will be produced:

www.test.com/main.asp,	**onclick**,	**div id = "next"**,	**text.innerHTML**,	**Updateinfo()**
↑	↑	↑	↑	↑
Dynamic page URL	*Event*	*Source*	*Target*	*Value*
www.test.com/main.asp,	**onclick**,	**div id = "previous"**,	**text.innerHTML**,	**Updateinfo()**
↑	↑	↑	↑	↑
Dynamic page URL	*Event*	*Source*	*Target*	*Value*

Second, multiple events may have the same triggering path that produces the same state. For instance, the following events produce the same state:

www.test.com/main.asp,	**onclick**,	**div id = "next"**,	**text.innerHTML**,	**Updateinfo()**
↑	↑	↑	↑	↑
Dynamic page URL	*Event*	*Source*	*Target*	*Value*
www.test.com/main.asp,	**onload**,	**div id = "next"**,	**text.innerHTML**,	**Updateinfo()**
↑	↑	↑	↑	↑
Dynamic page URL	*Event*	*Source*	*Target*	*Value*

To overcome this problem, the monitoring function has the ability of detecting and combining the information of all similar events in one field. This allows the WBC to

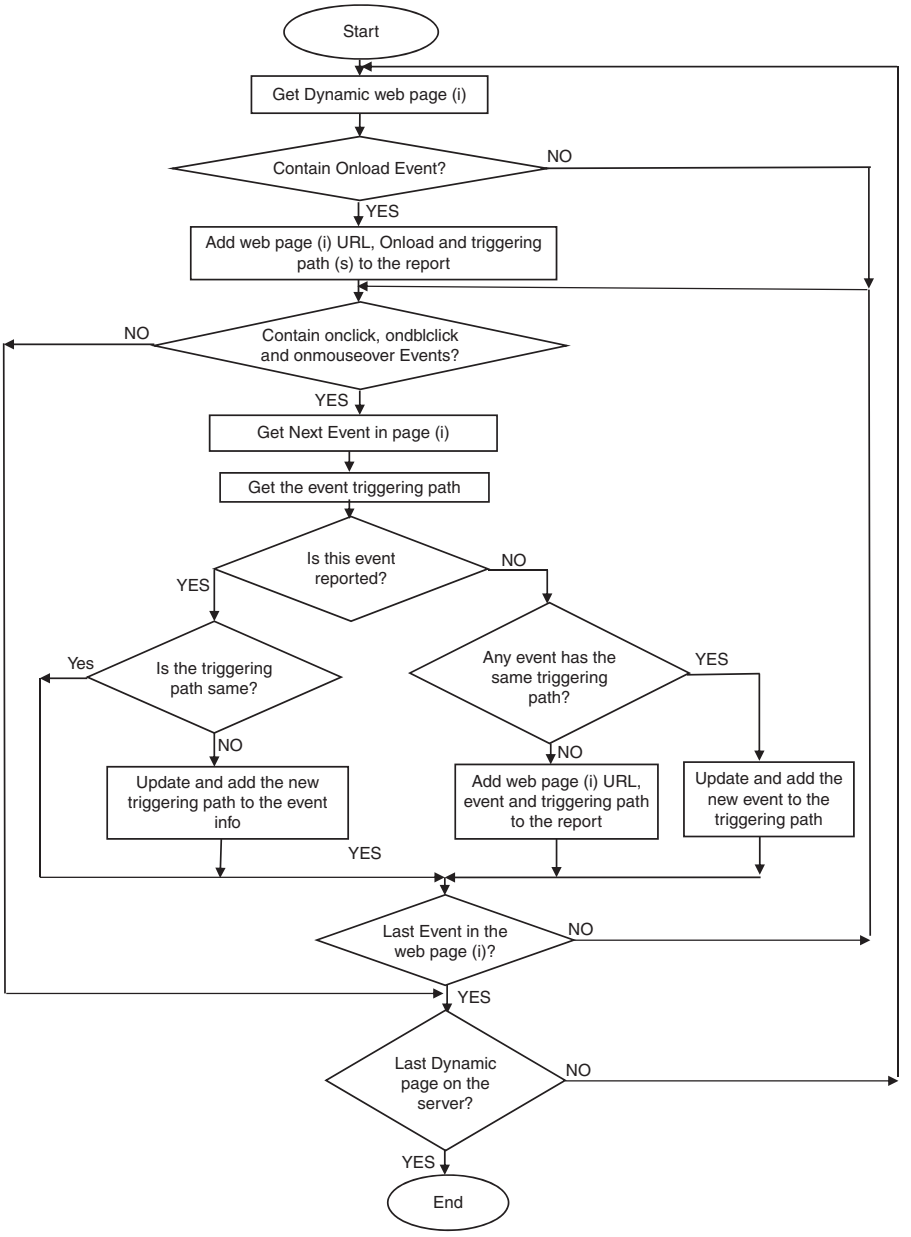


Figure 2.
Flowchart of the developed watcher file for adding the triggering paths of onload, onclick, ondblclick and onmouseover events to the report

trigger only one of the similar events. For instance, a group of combined similar events is represented in the report as follows:

www.test.com/main.asp, {**onclick**, **onload**}, {**button id** = "**move**", **div id** = "**next**"}, **text.innerHTML**, **Updateinfo**()

It is worth mentioning the followings: first, the watcher file will consider only the most important JavaScript events – onload, onclick, ondblclick and onmouseover (Duda *et al.*, 2009). Second, the WBC ignores the pages that require some database queries, which necessitates the user intervention to fill some forms, such as login pages. It is believed that this type of information is private and should not be indexed by crawlers.

(2) *Ranking function*: based on Figure 3, this function is responsible for ranking and adding the URLs of the updated and the new pages into the appropriate position of the watcher report. This allows the crawler to visit and download the most important pages first. In this paper, the rank of the web site pages is calculated according to the frequency of updating the page, and its closeness to the main page. An equal weight of 0.5 has been assigned to each of these factors. The calculation details of these factors are summarized below:

- *Frequency of updating the pages*: initially, the frequency value of all pages is set to 0. Whenever a page is updated, its frequency is incremented. Then, the frequency values will be normalized to be within the range {0, 0.5}. The normalized frequency of page i (Fn_i) is computed as:

$$Fn_i = \frac{1}{2}\left(\frac{F_i - F_{min}}{F_{max} - F_{min}}\right) \quad (1)$$

  where F_i is the frequency of updating the ith page, F_{min} and F_{max} are, respectively, the smallest and the largest assigned frequency value.

- *Closeness of the page to the root (main page)*: the main page and the pages that are linked to it get the highest level value, i.e., 0.5. All other pages are categorized and get a discrete level value of 0.1, 0.2, 0.3 or 0.4, depending on their closeness to the main page. The number of categories is specified by applying the rule of thumb (Chen *et al.*, 2012) on the length of the web site tree, i.e. no. of categories $= \sqrt{(length - 1)/2}$.

Finally, the rank of each page is calculated by adding its frequency and level values. It is worth mentioning that the watcher file saves and updates the frequency, the level and the rank values of each page in the report.

2.1.2 Watcher file setups. To run the watcher file on the web server, the administrator has to upload it to the main directory. Initially, the flag is reset to zero, and this allows the WBC to visit the web site. Then, if the web site is processed by the WBC, its flag will be set to one. This prevents other WBC copies to visit such web site. In the developed WBC, two strategies that reset the web site flag have been implemented. Furthermore, the search engine administrator has the ability to select one of these reset strategies. The details of these strategies are described below:

(1) *Event-based-reset strategy*: in this strategy, the watcher file reset the flag automatically when any of the web site pages is updated. This increases the chance

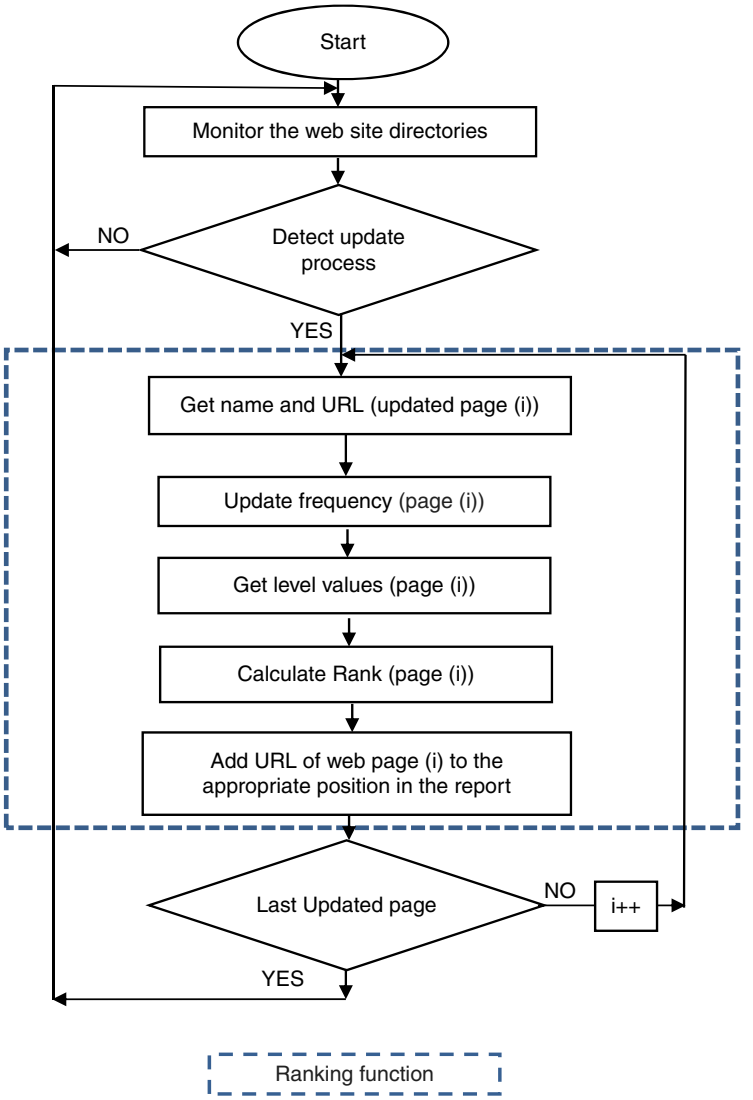


Figure 3.
Flowchart of the developed watcher file for adding the updated static pages to the report

of re-visiting this web site. On the other hand, visiting the same web site many times may lead to decreasing the chance of visiting other lower ranked web sites.

(2) *Time-based-reset strategy*: in this strategy, the watcher file reset the flag after a period of time which is assigned by the search engine administrator. The main advantage of this strategy is that all processed web sites will be visited once in a pre-defined period of time.

2.2 WBC-server design

To improve the performance of the developed WBC, the following schemes are used: first, multiple crawlers are run concurrently, where each crawler can run multiple

threads. Second, the WBC work has been divided into five units: controller unit; crawler unit; link extractor unit; AJAX unit; and link sorter unit, as shown in Figure 1. The functions of each unit are described below:

(1) *Controller unit*: this unit is responsible for the following:

- Specify the number of working crawlers, and the associated threads.
- Cache the DNS tables: in the case of connecting a crawler to a web site, it will contact with the DNS server to translate the web site domain name into IP address (Olston and Najork, 2010). As the DNS requests may require a large period of time, the controller unit of the proposed WBC is responsible for preparing and maintaining the WBC DNS caches. Hence, WBC crawler's URLs frontiers contain the IPs instead of the domain names, and this will speed up the crawling performance.
- Distributing the URLs in the main frontier(s) between the running crawlers. In addition, the controller has the ability of updating the working crawler's frontiers.

(2) *Crawler unit*: multiple copies of this unit can run concurrently, where each copy is responsible for visiting the IPs in its frontier, reads the watcher report and downloads the updated pages in a directory named as "LastDownloadedPages." To avoid degrading the performance of any visited server, the developed WBC is designed such that it allows only one copy of the crawler unit to visit the server at the same time. This crawler unit has the ability of processing static and dynamic web pages as described below:

- *Processing static pages*: the flowchart of the developed crawler unit for processing static web pages is shown in Figure 4. In the case of static pages, this crawler unit gets the URLs of the updated and the newly added pages from the report. Then, it will check the Robot.txt file, which includes downloading permissions and specifies the files to be excluded by the crawler (Kausar *et al.*, 2013). In the case that the crawler unit does not find the Robot.txt file, it will visit all updated and newly added web site pages.
- *Processing dynamic pages*: the process of indexing all dynamic pages requires a large period of time as the AJAX crawlers have to visit all web pages and search and trigger AJAX events to reach the dynamic information. In this paper, as explained in Section 2.1.1, the developed watcher file has been designed to detect and add the AJAX events including its triggering path to the watcher report. Hence, when the crawler visits the web site, it downloads the dynamic pages and its report to be processed by the AJAX unit. This way decreases the processing time which improves the crawling efficiency.

(3) *Link extractor unit*: this unit is responsible for getting the pages from the "LastDownLoadedPages" directory, extracts and saves the pages URLs in a queue named as "ExtractedURLsQueue." In this unit, the following are executed:

- Remove the processed pages from the "LastDownLoadedPages" directory and save the static pages in the database. In addition, it moves the dynamic pages to a directory named as "LastDownloadedAJAX Pages" to be processed by the AJAX unit.

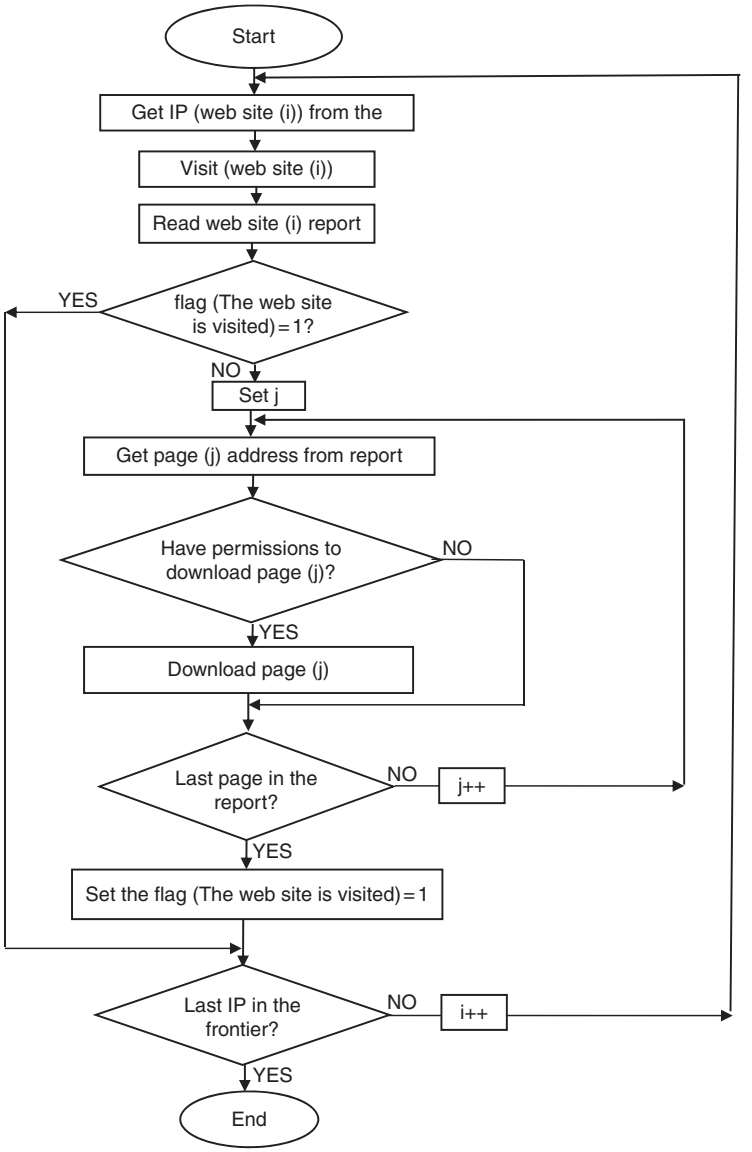


Figure 4.
Flowchart of the developed crawler unit for processing static web pages

- In order to obey the webmaster restriction, the link extractor discards all URLs whose attribute "rel" set to "nofollow."
- Add the URL of the web site main page only to the "ExtractedURLsQueue." This is because the watcher report provides the crawler the URLs of the updated pages. This is done by abstracting the main web site URL for each of the extracted URLs. For example, all "cmpe.emu.edu.tr" web site pages such as cmpe.emu.edu.tr/FacultyMemberList.aspx are abstracted to cmpe.emu.

edu.tr. It is worth mentioning that most of the current crawler approaches extract all the URLs of the visited pages, and these URLs are visited and processed as well. This process may consume the system resources such as the memory and CPU cycles, etc. In addition, the crawler has to request a connection to a web server whenever it has a URL of a page hosted on it.

- This unit removes the duplicated URLs as well.

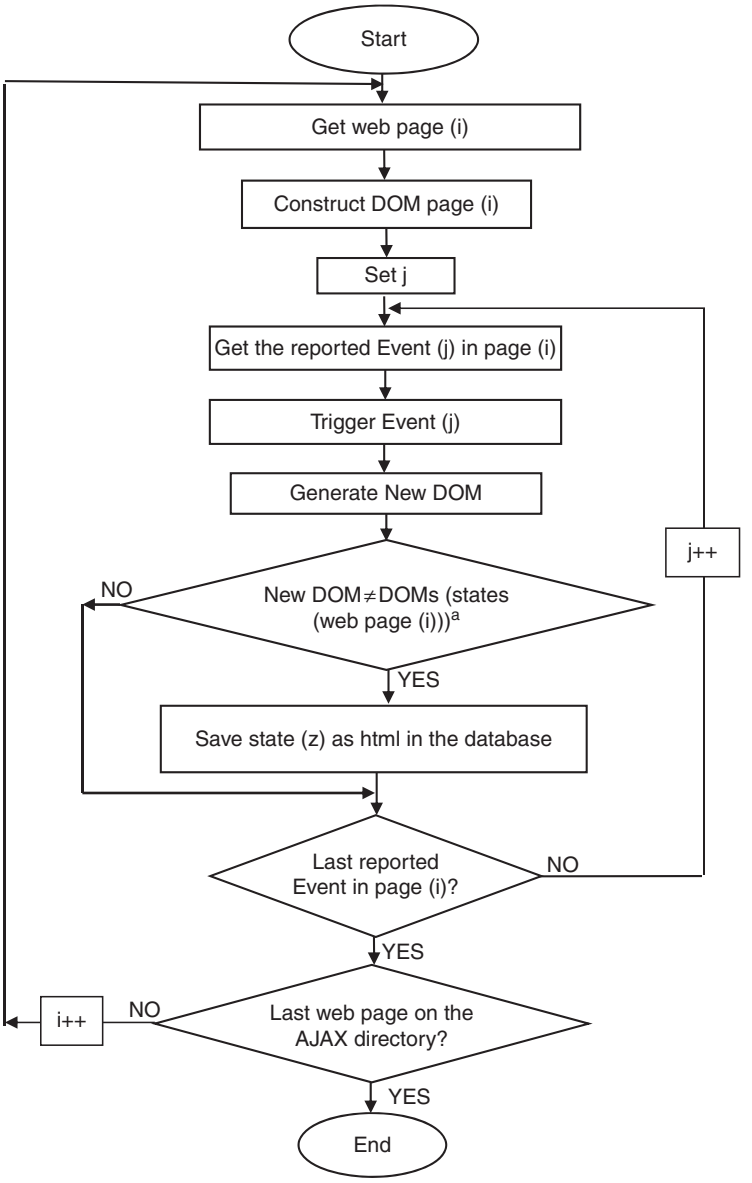
(4) *AJAX unit*: this unit is responsible for getting the dynamic pages from the "Last DownloadedAJAXPages" directory and triggering the events that have been reported by the watcher file instead of triggering all pages events. Figure 5 shows the flowchart for processing the dynamic pages by the AJAX unit. To detect new states, the DOM of the generated state is compared with the DOMs of the previously founded states. This is done by applying Algorithm 1.

Algorithm 1: detecting the AJAX pages distinct states

1: Reset **flag = false**; // The variable flag is set to true if the generated state is new.
// NGD refers to the new generated DOM.
2: Remove the information of useless and irrelevant tags form the NGD
3: **For i** = 1 **to** N **do** // N: Number of the previously founded states.
4: **For j** = 1 **to** M **do** // M: Number of elements (tags) in the generated state.
5: **If** ($E^{J}_{DOM\ (i)}$.Contents != E^{J}_{NGD}.Contents**) then** // E is an element (tag) in DOM(I) and NGD
6: **flag** = **true;**
7: **Break;**
8: **ELSE**
9: **flag** = **false;**
10: **End If**
11: **End For**
12: **If (flag** = = **false) then** //The NGD and DOM (i) are identical
13: **Exit;**
14: **End If**
15: **End For**
16: **If (flag** = = **true) then**
17: **Set NGD as one of the page state**
18: **Save the html of NGD in the database**
19: **End If**

It is worth mentioned that, webpages may contain useless and irrelevant information for crawling, such as advertisements, timestamps and counters, which are changed very frequently (Choudhary *et al.*, 2012) and to insure that such information will not mislead the comparison process, the AJAX unit deletes these tags from the generated state, as mentioned in Algorithm 1, Step 2. Finally, in order to obtain the dynamic content, the JavaScript engine V8 (2014), and the embedded web browser Chromium (2014), are used by this unit to parse the JavaScript code in a web page, trigger its reported events and constricting the new DOMs.

(5) *Link sorter unit*: this unit is responsible for getting the URLs from the "ExtractedURLsQueue" and adds those URLs to the appropriate position in the main URLs frontier(s) based on the sorting process. In this paper, the back link



Note: [a]This process is done according to Algorithm 1

Figure 5. Flowchart for processing the dynamic pages by the AJAX unit

count algorithm has been adopted to sort the URLs (Ward and French, 2013; Kelly and Nixon, 2013). Moreover, the link sorter unit has the ability of using a group of main frontier. In such case, each frontier has a rank depending on the importance and the rank of its URLs. Hence, the URLs of most important frontier (s) are visited faster and more frequently than the URLs of the lower ranked frontier(s).

2.3 WBC properties

The developed WBC has the following properties:

(1) Platform independence: the watcher file can be added to and work under any platform.

(2) Low cost and high performance: the watcher file increases the number of visited web sites at almost no cost. The web site administrator can get the watcher for free. In addition, an open source demo will be available for research purposes.

(3) Parallel independent crawlers: many independent crawlers can work in parallel with multiple threads.

(4) The watcher file has the ability to perform its duties on the hosting servers, i.e., one copy of the watcher file can monitor a group of web sites that are hosted on the same server.

(5) The running crawlers can only read the watcher report and have no permission to control or contact the watcher file itself. This feature protects the web sites servers and its data from any possibility of violation or attacked by spams softwares through using the watcher file.

(6) Failure recovery: in the case that the controller unit did not receive information from any working crawler, the controller will consider that crawler as a dead one and the IPs in its frontier will be assigned to a new crawler.

(7) Dynamicity of the assignment function: the controller unit has the ability to analyze previous work of the crawler and build statistic reports that helps in estimating the number of required crawlers and balancing the distribution of URLs. In addition, if any frontier contains a large number of IPs, the controller can run new crawlers and re-divides the IPs. Hence, the time required to visit all IPs is decreased.

(8) By making use of the watcher report that provides the crawler the URLs of the updated pages, the "ExtractedURLsQueue" will contain only the main web sites URL. This process decreases the number of URLs in the "ExtractedURLsQueue" and in the main frontier(s). This not only save the system resources but also significantly decreases the number of connection requests to each web server.

(9) Flexibility of the assignment function: by making use of the flag, that has been introduced for solving the overlapping problem, the proposed WBC can work with any assignment function.

(10) To avoid degrading the performance of the WBC server, if any of the directory or queue of the server units is found to be empty, the corresponding unit will be set to inactive mode, and will be re-activated when new data are added to its corresponding path.

(11) The watcher file has the ability to set the services of the ranking function to inactive mode during the server rush times, and later re-activate these services. This feature avoids degrading the performance of the web site server.

(12) Unlike, sitemap crawler, the proposed WBC perform all crawling processes in the sense that it detects all updated and newly added pages automatically without any human explicit intervention or downloading the entire web sites.

3. Experimental study

In the following experiments, the number of days and the number of samples used in this study have been selected to achieve a 95 percent confidence level. In addition, all of the experiments were conducted using the time-based-flag reset mechanism. The server PC used in this study has the following characteristics: Intel Core i7 CPU; 2.4 GHz clock frequency; 8 GB RAM; 1 TB hard drive; and the operating system was Windows 7.

The performance of the conventional crawler, and the proposed WBC for static and dynamic web sites was studied in seven experiments. In the first experiment, the number of updated web pages in a group of web sites was investigated. The crawlers overlapping and communication problems were studied in the second experiment. In the third experiment, the effects of the watcher file on the web sites server have been investigated. The number of downloaded distinct web pages using the WBC and crawlers of Mukhopadhyay *et al.* (2006), Apache Nutch (2015) and Scrapy (2015) was examined in the fourth experiment. In the fifth experiment, the performance of the re-visiting policies was investigated. Finally, the WBC performance for processing AJAX web sites and AJAX real data set were studied in experiments six and seven, respectively.

3.1 Experiment 1: percentage of updated web pages in a group of web sites

In this experiment, the watcher file has been used to monitor a group of web sites that belong to different categories, such as education, sport, banks and news, for one week and the number of updated pages in each web site was recorded and shown in Table I. The average percentage of the updated and the newly added pages was calculated using the following equation:

$$\text{Average } \% = \frac{1}{7}\sum_{D=1}^{7}\frac{N(D)}{M} \times 100\% \tag{2}$$

where N represents the number of updated and newly added web pages in a specific day, M represents total number of web pages. This experiment was performed by downloading a copy of the monitored web sites pages. Then, a copy of the watcher file was run for each web site to detect the updated and the newly added pages. The following can be seen from Table I: first, less than 12 percent

Web site no.	Web site URL	Total number of web pages (M)	Number of updated and newly added web pages (n)							Average %
			Day1	Day2	Day3	Day4	Day5	Day6	Day7	
1	cmpe.emu.edu.tr	≈591	74	70	60	77	65	63	62	≈11.5
2	ahu.edu.jo/colleges/engineering	≈402	43	44	40	50	35	39	34	≈10
3	global.nytimes.com	≈15,090	1,346	1,403	1,321	1,297	1,273	1,432	1,357	≈9
4	garanti.com.tr/en	≈153	10	8	11	10	7	9	8	≈6
5	picosoftengineering.com	≈35	2	1	2	2	4	3	2	≈5
6	Sarayanews.com	≈3,377	80	110	89	120	81	94	85	≈3
7	Kooora.com	≈9,320	200	150	170	220	190	175	195	≈2

Table I. The number of updated pages for different web sites recorded in seven days

of the web pages are updated. This means that the conventional crawling techniques are spending most of their working time in visiting the un-updated pages. Second, the percentage of the updated web pages in web sites four to seven was very small. This is due to the fact that such web sites have to keep the old information available to the users, and the update process is done by adding new web pages.

3.2 Experiment 2: crawler overlapping and communication problems study

In this experiment, the crawler overlapping and communication problems were studied for the developed WBC and for the following popular three crawling scenarios, which we have re-implemented: first, independent crawlers, which means that the working crawlers have no communication between each other and more than one crawler may visit the same web site. Second, dependent crawlers, which means that the crawlers have to communicate with each other before visiting a new web page. Third, seed-server crawler: this scenario eliminates the communication between the running crawlers by introducing a seed-server which is responsible for crawler's decisions such as allowing a crawler to visit a specific web site (Mukhopadhyay *et al.*, 2006).

The experiment was conducted by running 50 parallel copies for each scenario for three days. In addition, the crawlers were allowed to crawl all web sites under the domain of "emu.edu.tr" and the extracted outer links (if any). Table II shows the number of times that randomly selected web pages were downloaded for the four scenarios, and Figure 6 shows the number of downloaded web pages. From Table II and Figure 6, the followings can be derived:

(1) In the independent scenario, more than 40 out of 50 crawlers have visited the same web pages. This means that the percentage of the overlapping problem was above 80 percent, and this decreases the performance of the crawlers. On the other hand, these crawlers were able to download on the average 688,120 web pages.

(2) Although the dependent scenario has reduced the overlapping problem (less than seven out of 50 crawlers have visited the same web pages), the average of total number of distinct downloaded web pages was less than 240 thousands.

(3) The seed-server strategy has successfully solved the overlapping problem. On the other hand, this approach was able to download only around 350,816 distinct web pages.

Table II.
The frequency of downloading specific web pages in three days by running 50 parallel crawlers

	Frequency											
	Independent			Dependent			Seed-server			WBC		
Web page no.	Day1	Day2	Day3	Day1	Day2	Day3	Day1	Day2	Day3	Day1	Day2	Day3
Web page (1)	43	44	47	5	7	4	2	1	1	1	0	0
Web page (2)	44	41	43	4	3	2	1	2	2	1	1	1
Web page (3)	42	42	45	7	5	4	2	1	2	1	1	1
Web page (4)	47	45	43	1	7	3	1	2	1	1	0	0
Web page (5)	44	43	40	7	5	4	1	2	3	1	0	0

(4) Finally, by making use of the flag in the developed WBC there is no communication between the running crawlers, and only one crawler can process a specific page. In addition, the WBC was able to download approximately one million distinct web pages, which is almost double the number of the downloaded pages as compared with the independent strategy, which was found to be the second best scenario. Furthermore, the WBC has downloaded the five pages in the first day, as shown in Table II, and then downloads only the updated pages in the second and third day. Finally, it must be mentioned that the WBC was the only approach which is capable of downloading the updated pages only, while the other scenarios may download updated and un-updated pages.

3.3 Experiment 3: watcher file effects on the site servers

To investigate the effects of the watcher file on the web sites server, the following experiment has been conducted. In this experiment, the size of the allocated memory and the percentage of CPU usage for the watcher file, when it is uploaded and run on a server that contains 10,000; 100,000; and 1,000,000 web pages have been recorded and shown in Table III.

It is clear from Table III that the watcher file does not require large CPU and memory resources. For instance, only 3.45 percent of CPU usage and around 197 KB of the memory were used for 1,000,000 web pages.

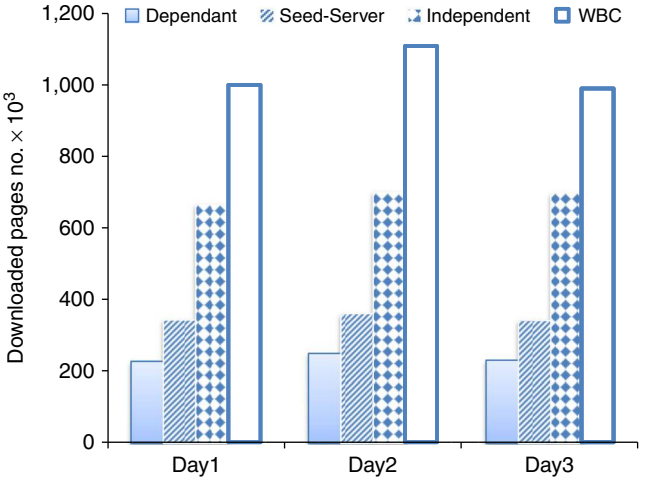


Figure 6.
The total number of downloaded web pages using dependent, seed-server, independent strategies, and the developed WBC in three days

Number of web pages	Size of allocated memory (KB)	% of CPU usage
10,000	62.351	0.84
100,000	155.857	2.37
1,000,000	197.140	3.45

Table III.
The watcher file requirements

3.4 Experiment 4: number of distinct downloaded web pages study

In the first part of this experiment, the performance of the proposed WBC and the WEB-SAILOR crawler (Mukhopadhyay *et al.*, 2006), was studied. We have re-implemented the same scenario of Mukhopadhyay *et al.* (2006). As the SAILOR crawler requires a seed-server, three machines were used for this approach while two machines were used for the proposed WBC. Two machines were used as crawler clients for both scenarios: one for .com domain; and the other for downloading the pages of .edu, .net and .org. In addition, due to the huge number of published .com web sites compared with other domains, the number of threads for the crawler clients has been specified to be 25 for .com, and 10 for .edu, .net and .org (Mukhopadhyay *et al.*, 2006; Netcraft, 2014). Finally, both crawlers were allowed to crawl a group of randomly selected URLs, and then visit all extracted URLs. The number of distinct web pages that the two approaches were able to download was recorded in Table IV. It can be seen from Table IV that the proposed WBC increases the number of visited web sites by approximately a factor of three as compared with the SAILOR crawler.

The performance of the proposed WBC was also compared with the most popular open source crawlers: Apache Nutch (2015); and Scrapy (2015). This experiment was repeated for seven days where one copy of Apache Nutch, Scrapy and WBC was run on a separate machine, and each crawler can run at most 50 threads. The crawlers were allowed to crawl all web sites under the domain of "emu.edu.tr" and the extracted outer links (if any). The number of web pages that have been processed by these crawlers were recorded in Table V. It is clear from Table V that the proposed WBC outperforms both the Apache Nutch, and Scrapy crawlers by a factor of approximately 1.6, and 1.4, respectively.

Table IV.
Number of downloaded distinct web pages using the proposed WBC and WEB-SAILOR

Day	Number of downloaded web pages	
	WBC	WEB-SAILOR
Day 1	381,007	126,981
Day 2	390,074	131,045
Day 3	383,941	126,987

Table V.
Number of downloaded web pages using the proposed WBC, Apache Nutch, and Scrapy

Day	Number of downloaded web pages		
	WBC	Apache Nutch	Scrapy
Day 1	113,854	71,154	82,140
Day 2	114,978	70,447	80,261
Day 3	114,864	71,601	81,126
Day 4	115,912	72,008	82,307
Day 5	114,991	70,397	81,151
Day 6	114,784	71,029	81,353
Day 7	115,005	72,042	82,046

3.5 Experiment 5: the performance of the re-visiting policies

In this experiment, the re-visiting performance has been investigated for the WBC and the following three re-visiting policies: the uniform (Pichler *et al.*, 2011; Bhute and Meshram, 2010; Leng *et al.*, 2011; Sharma *et al.*, 2012; Singh and Vikasn, 2014); the proportional by rank (Pichler *et al.*, 2011; Bhute and Meshram, 2010; Leng *et al.*, 2011); and the proportional by top *N* levels (Pichler *et al.*, 2011; Cho *et al.*, 2012). This experiment was repeated for seven days where one crawler with five threads was used for each policy. In addition, the crawlers were responsible to visit and process news web site: www.hkjtoday.com. The performance of the re-visiting policies was studied according to the following factors: first, the time required for re-visiting and downloading a web site pages; and second, the percentage of the updated pages that the crawler was able to download at the time of visiting. In the proportional policies, the number of levels (*N*) and the threshold values were selected based on experimental values that provide the best result. The required time for processing the web site and the percentage of downloading the updated and newly added pages are shown in Figures 7 and 8, respectively.

It can be seen from Figure 7 that the proposed WBC decreases the web site processing time by a factor of 2.5 as compared with the second best scheme, which is found to be the proportional by ranking. In addition, the processing time for the uniform policy is approximately ten times that of the WBC. This is because the uniform policy has to visits and downloads the entire web site. Furthermore, it can be seen from Figure 8 that the WBC and the uniform policy were able to download all the updated pages in most cases, whereas the proportional policies were able to download at most 82 percent of the updated pages. Based on this study, it is clear that the WBC outperform the performance of the other re-visiting policies.

3.6 Experiment 6: WBC performance for processing AJAX web sites

In this experiment, the performance of the proposed WBC and the crawler developed in Cui *et al.* (2013) for processing AJAX web sites were studied. The same scenario performed in Cui *et al.* (2013) has been repeated by using the WBC with 1, 2, 3, and 4

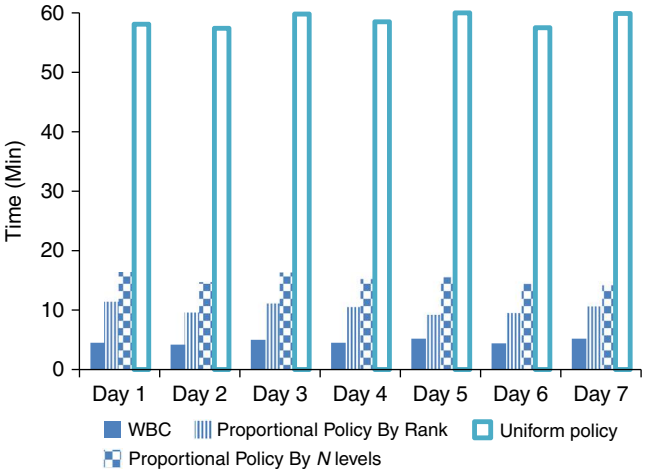


Figure 7.
The required time for re-visiting "www.hkjtoday.com" web site using the crawlers of WBC, uniform, and proportional by rank and by top *N* level

crawlers, where each of the working crawlers can run up to four threads. The number of processed dynamic pages in ten minutes was recorded and shown in Figure 9. It can be seen from Figure 9 that the proposed WBC increases the number of the processed dynamic pages by approximately 20 percent as compared with the approach of Cui *et al.* (2013).

3.7 Experiment 7: WBC performance for real AJAX data set

In this experiment, the performance of the proposed WBC for processing a real AJAX data set was studied. This experiment was performed on YouTube videos data set (Cheng *et al.*, 2008; available at: http://netsg.cs.sfu.ca/youtubedata/). The main aim of this experiment is to investigate the ability of the WBC to access and download the related comment pages written by the video viewer. The number of the video comments reported by Cheng *et al.* (2008), and the number of comments that the WBC was able to obtain by triggering the related AJAX events are recorded in Table VI. It can be seen from Table VI that the proposed WBC was able to download more than 98 percent of the reported AJAX comment pages.

4. Conclusions

In this paper, a WBC has been presented to improve the overall performance of search engines. The developed WBC visits only the updated and the newly added pages in each web site. The proposed WBC has the ability of crawling dynamic as well as static pages. In addition, it allows solving the crawlers overlapping problem and eliminates the need of communication between running crawlers. Several experiments have been conducted and it has been observed that the proposed WBC significantly increases the number of visited web sites, and decreases the web sites re-visiting time as compared with other popular approaches. Moreover, the proposed WBC increases the number of processed dynamic pages by approximately 20 percent as compared with the approach of Cui *et al.* (2013). Finally, the WBC can be extended by integrating other RIAs techniques such as VBScript and JavaFX, and can be used for improving the performance of topic specific crawlers which is left as a future work.

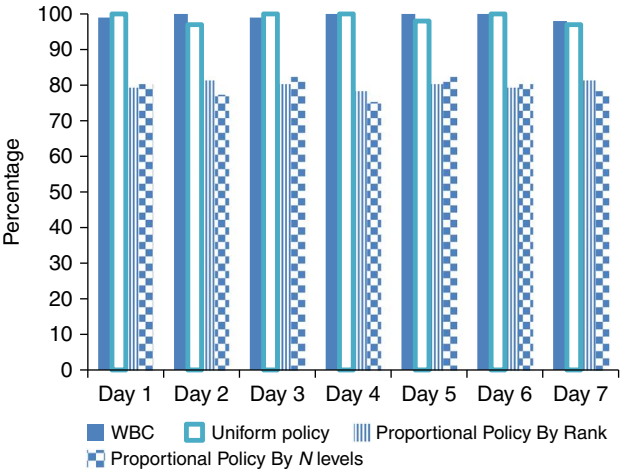


Figure 8.
The percentage of downloading updated pages in "www.hkjtoday.com" web site using the crawlers of WBC, uniform, and proportional by rank and by top *N* levels

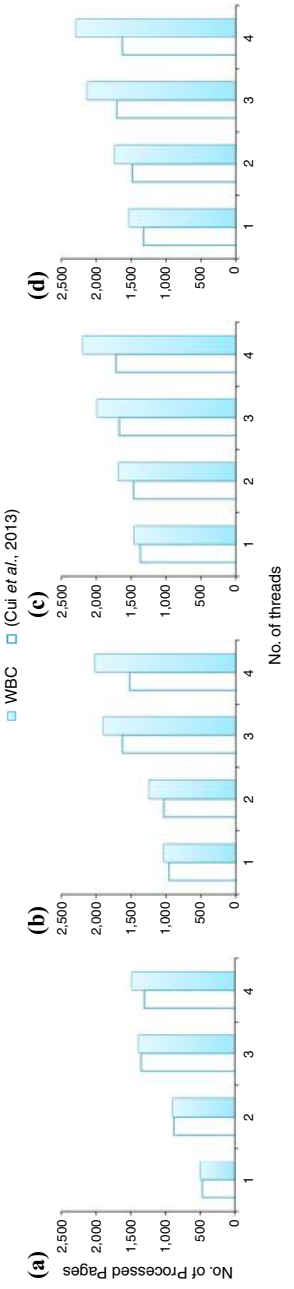


Figure 9.
Number of dynamic pages processed in ten minutes for (a) one crawler, (b) two crawlers, (c) three crawlers, and (d) four crawlers

Table VI.
Number of comments for specific YouTube videos reported by Cheng *et al.* (2008) and downloaded using the proposed WBC

Video name	Number of comments Reported in Cheng *et al.* (2008)	Downloaded by the WBC	% of the downloaded comments
New Numa – The Return of Gary Brolsma!	17,657	17,589	99.61
CRAZED NUMA FAN !!!!	3,010	3,010	100
Influence	3,866	3,862	99.90
My United States of […] WHATEVA !!!	4,447	4,440	99.84
Heather Martin – When Are You Coming Home	6,045	6,044	99.98
The COMMENT Video!!	976	960	98.36
iBlinds – New generation iPod accessory	3,606	3,606	100
Smosh Short 1: Dolls	7,919	7,875	99.44

References

Agarwal, A., Singh, D., Pandey, A.K.A. and Goel, V. (2012), "Design of a parallel migrating web crawler", *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 2 No. 4, pp. 147-153.

Amin, M.R., Prince, M.A. and Hussain, M.A. (2012), "WEBTracker: a web crawler for maximizing bandwidth utilization", *Journal of Science and Technology*, Vol. 16 No. 2, pp. 32-40.

Amolochitis, E., Christou, I.T., Tan, Z.H. and Prasad, R. (2013), "A heuristic hierarchical scheme for academic search and retrieval", *Information Processing & Management*, Vol. 49 No. 6, pp. 1326-1343.

Apache Nutch (2015), "Nutch 2.3", available at: www.nutch.apache.org/ (accessed May 20, 2015).

Bhushan, B., Gupta, M. and Gupta, G. (2012), "Increasing the efficiency of crawler using customized sitemap", *International Journal of Computing and Business Research*, Vol. 3 No. 2.

Bhute, A.N. and Meshram, B.B. (2010), "Intelligent web agent for search engines", *International Conference on Trends and Advances in Computational Engineering (TRACE – 2010), Barkatullah University, Bhopal*, pp. 211-218.

Brawer, S.B., Ibel, M., Keller, R.M. and Shivakumar, N. (2013), "Web crawler scheduler that utilizes sitemaps from websites", US Patent Application No. 13/858,872, available at: www.google.com/patents/US7769742/ (accessed September 9, 2015).

Bruno, E.J. (2006), "Ajax: asynchronous JavaScript and XML", *Dr Dobbs Journal*, Vol. 31 No. 2, pp. 32-35.

Chen, Y., Sanghavi, S. and Xu, H. (2012), "Clustering sparse graphs", *Advances in Neural Information Processing Systems 25 (NIPS 2012), Neural Information Processing Systems*, pp. 2204-2212.

Cheng, X., Dale, C. and Liu, J. (2008), "Statistics and social network of YouTube videos", IEEE, 16th International Workshop on Quality of Service, University of Twente, pp. 229-238.

Cho, J., Garcia-Molina, H. and Page, L. (2012), "Reprint of: efficient crawling through URL ordering", *Computer Networks*, Vol. 56 No. 18, pp. 3849-3858.

Choudhary, S., Dincturk, M.E., Mirtaheri, S.M., Moosavi, A., von Bochmann, G., Jourdan, G.V. and Onut, I.V. (2012), "Crawling rich internet applications: the state of the art", *Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012), IBM Corp*, pp. 146-160.

Chromium (2014), "Chromium 1.0.154.65", available at: www.chromium.org/Home/ (accessed April 7, 2014).

Cui, L.J., He, H. and Xuan, H.W. (2013), "Analysis and implementation of an Ajax-enabled web crawler", *International Journal of Future Generation Communication and Networking*, Vol. 6 No. 2, pp. 139-146.

Cui, L.J., He, H., Xuan, H.W. and Li, J.G. (2013), "Design and development of an Ajax web crawler", *The Tenth International Conference on Computability and Complexity in Analysis, CCA Net*, pp. 6-10.

Dincturk, M.E., Jourdan, G.V., Bochmann, G.V. and Onut, I.V. (2014), "A model-based approach for crawling rich internet applications", *ACM Transactions on the Web (TWEB)*, Vol. 8 No. 3, pp. 1-19.

Duda, C., Frey, G., Kossmann, D., Matter, R. and Zhou, C. (2009), "Ajax crawl: making Ajax applications searchable", *25th IEEE International Conference on Data Engineering*, pp. 78-89.

Kausar, M.A., Dhaka, V.S. and Singh, S.K. (2013), "Web crawler – a review", *International Journal of Computer Applications*, Vol. 63 No. 2, pp. 31-36.

Kelly, B. and Nixon, W. (2013), "SEO analysis of institutional repositories: what's the back story?", Open Repositories 2013, 2013-07-08-2013-07-12, Charlestown.

Kumar, M.S. and Neelima, P. (2011), "Design and implementation of scalable, fully distributed web crawler for a web search engine", *International Journal of Computer Applications*, Vol. 15 No. 7, pp. 8-13.

Leng, A.G.K., Ravi, K.P., Singh, A.K. and Dash, R.K. (2011), "PyBot: an algorithm for web crawling", *International Conference on Nanoscience Technology and Societal Implications (NSTSI -2011), IEEE, CV Raman College of Engineering*, pp. 1-6.

Mishra, S., Jain, A. and Sachan, A.K. (2010), "Smart approach to reduce the web crawling traffic of existing system using HTML based update file at web server", *International Journal of Computer Applications*, Vol. 11 No. 7, pp. 34-38.

Mukhopadhyay, D. and Sinha, S. (2008), "A new approach to design graph based search engine for multiple domains using different ontologies", *International Conference on Information Technology (ICIT '08), IEEE, Bhubaneswar*, pp. 267 -272.

Mukhopadhyay, D., Mukherjee, S., Ghosh, S., Kar, S. and Kim, Y.C. (2006), "Architecture of a scalable dynamic parallel WebCrawler with high speed downloadable capability for a web search engine", *The 6th International Workshop MSPT 2006 Proceedings, Youngil Publication*, pp. 103-108.

Nath, R. and Bal, S. (2011), "A novel mobile crawler system based on filtering off non-modified pages for reducing load on the network", *The International Arab Journal of Information Technology*, Vol. 8 No. 3, pp. 272-279.

Netcraft (2014), "Web server survey", available at: http://news.netcraft.com/archives/2014/01/03/january-2014-web-server-survey.html/ (accessed July 4, 2014).

Olston, C. and Najork, M. (2010), "Web crawling", *Foundations and Trends in Information Retrieval*, Vol. 4 No. 3, pp. 175-246.

Pichler, C., Holzmann, T. and Wright, B. (2011), "Information search and retrieval", Crawler Approaches and Technology, available at: www.iicm.tugraz.at/0x811bc82b_0x0011b4d6/ (accessed September 9, 2015).

Schonfeld, U. and Shivakumar, N. (2009), "Sitemaps: above and beyond the crawl of duty", *18th International Conference on World Wide Web, ACM*, pp. 991-1000.

Scrapy (2015), "Scrapy 1.0", available at: www.scrapy.org (accessed May 20, 2015).

Sharma, A.K., Gupta, J.P. and Agarwal, D.P. (2010), "Parcahyd: an architecture of a parallel crawler based on augmented hypertext documents", *International Journal of Advancements in Technology*, Vol. 1 No. 2, pp. 270-283.

Sharma, V., Kumar, M. and Vig, R. (2012), "A hybrid revisit policy for web search", *Journal of Advances in Information Technology*, Vol. 3 No. 1, pp. 36-47.

Singh, A.V. and Vikas, A.M. (2014), "A review of web crawler algorithms", *International Journal of Computer Science & Information Technologies*, Vol. 5 No. 5, pp. 6689-6691.

Uzun, E., Agun, H.V. and Yerlikaya, T. (2013), "A hybrid approach for extracting informative content from webpages", *Information Processing & Management*, Vol. 49 No. 4, pp. 928-944.

V8 (2014), "V8 3.31.1", available at: https://developers.google.com/v8/ (accessed April 7, 2014).

Ward, E. and French, G. (2013), *Ultimate Guide to Link Building: How to Build Backlinks. Authority and Credibility for Your Website, and Increase Click Traffic and Search Ranking*, Entrepreneur Press, CA.

Wu, M. and Lai, J. (2010), "The research and implementation of parallel web crawler in cluster", *International Conference on Computational and Information Sciences (ICCIS), IEEE, Chengdu*, pp. 704-708.

Yang, Y., Du, Y., Hai, Y. and Gao, Z. (2009), "A topic-specific web crawler with web page hierarchy based on HTML dom-tree", *Asia-Pacific Conference on Information Processing (APCIP 2009), IEEE, Shenzhen*, pp. 420-423.

Yao, Z., Daling, W., Shi, F., Yifei, Z. and Fangling, L. (2012), "An approach for crawling dynamic webpages based on script language analysis", *Ninth Web Information Systems and Applications Conference (WISA), IEEE, Haikou*, pp. 35-38.

Corresponding author
Dr Saed Alqaraleh can be contacted at: saed.alqaraleh@emu.edu.tr